REMARKS/ ARGUMENTS

The Office Action of November 30, 2004 has been carefully reviewed and this response addresses the Examiner's concerns.

Status of the Claims

Claims 1-25 were pending in this application.

Claims 2 is cancelled without prejudice.

Claims 20-25 were rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

Claims 20-25 were rejected under 35 U.S.C. 101 because the claimed invention is the disclosed invention is inoperative and therefore lacks utility.

Claims 1, 5, 8-13, 15-16 and 19 were rejected under 35 U.S.C. 102(e) as being anticipated by Buzsaki.

Claims 2-4, 14 and 20-23 were rejected under 35 U.S.C. 1 03(a) as being unpatentable over Buzsaki in view of Winokur et al.

Claims 6, 7, 17, and 18 are allowed.

Claims 1, 3, 13, 20-25 are amended.

Support for Amendments to the Claims

The amendments to the claims do not raise new issues or require additional search. The amendments utilize language used in the previously presented claims and, therefore, should not require an additional search.

The 35 U.S.C. §101 rejection

Claims 20-25 were rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

Claims 20-25 were rejected under 35 U.S.C. 101 because the claimed invention is the disclosed invention is inoperative and therefore lacks utility.

Claims 20-25 are amended in order to place them in Beauregard form. In *Beauregard,* Gary M. Beauregard et al. appealed the Board of Patent Appeals and Interferences decision rejecting computer program product claims as being non statutory. Since their appeal followed the *In re Lowry* decision (*In re Lowry,* 32 F. 3d 1579 (Fed. Cir. 1994).), the Commissioner stated that "computer programs embodied in a tangible medium, such as floppy diskettes, are patentable subject matter under 35 U.S.C. § 101 and must be examined under 35 U.S.C. §§ 102 and 103." *In re Beauregard,* 32 F. 3d 1583 (Fed. Cir. 1994). *In re Beauregard* is still good law. Therefore, Applicants respectfully state that claims 20 through 25 claim patentable subject matter under 35 U.S.C. § 101.

<u>The 35 U.S.C. §102 rejections</u>

Claims 1, 5, 8-13, 15-16 and 19 were rejected under 35 U.S.C. 102(e) as being anticipated by Buzsaki.

Independent claims 1 and 13 are amended to claim a management system <u>managing at least one network element of a communication network</u>. The Examiner states, in examining claim 2, that Buzsaki (the '193 patent) does not explicitly teach "the managed system being a network element of a communication network." Since the amended claim 1 incorporates the limitations of claim 2, Buzsaki does not anticipate the amended claim.

Since amended independent claims 1 and 13 include the limitations of previously submitted claim 2 (now canceled) and some of the limitations of claim 14, amended claims 1 and 13 and their dependent claims are considered under the remarks for the 35 U.S.C. §103 rejection.

<u>The 35 U.S.C. §103 rejections</u>

Claims 2-4, 14 and 20-23 were rejected under 35 U.S.C. 1 03(a) as being unpatentable over Buzsaki in view of Winokur et al.

Since, as stated above, amended independent claims 1 and 13 include the limitations of previously submitted claim 2 (now canceled) and some of the limitations of claim 14, amended claims 1 and 13 and their dependent claims are considered below.

Amended claim 1 claims a method for defining a management policy for controlling behavior of a management system, where the management system manages at least one network element of a communication network, said method comprising:

executing a program on a processor-based device that presents a user interface for defining said management policy;

receiving input from a user identifying management action to be performed by said management policy; and

receiving input from a user specifying a modifiable process flow for said management policy to utilize in performing said management action.

Amended claim 13 claims a management system managing at least one network element of a communication network, the management system comprising:

a software program stored to a data storage device, said software program executable to present a user interface for defining a management policy for controlling behavior of said management system;

at least one processor-based device operable to execute said software program; and

at least one input device communicatively coupled to said at least one processor-based device to allow input from a user to said software program to identify management action to be performed by said management policy and to specify a modifiable process flow for said management policy to utilize in performing said management action.

Central to each of these claims is the notion of a management policy used for managing at least one network element of a communication network.

The first step in determining whether a claim is anticipated, or is obvious in view of prior art, is to interpret the claim. ("It is elementary in patent law that, in determining whether a patent is valid ...., the first step is to determine the meaning and scope of each claim in suit." *Lemelson v. Gen. Mills, Inc.*, 968 F.2d 1202, 1206, 23 U.S.P.Q.2D (BNA) 1284, 1287 (Fed. Cir. 1992).) When not defined by applicant in the specification, the words of a claim must be read as they would be interpreted by those of ordinary skill in the art. (MPEP 211.01) (*Rexnord Corp. v. Laitram Corp.*, 274 F.3d 1336, 1342, 60 USPQ2d 1851, 1854 (Fed. Cir. 2001) ("explaining the court's analytical process for determining the meaning of disputed claim terms")).

The term "management policy" has a well-defined meaning in terms of managing at least one network element of a communication network. For example, in his 1994 paper, Sloman defined management policy as "the information which influences the interactions between a **subject** and a **target** and so the policy specifies a relationship between the subject and target." (Morris Sloman, **POLICY DRIVEN MANAGEMENT FOR DISTRIBUTED SYSTEMS,** Journal of Network and Systems Management, Plenum Press. Vol.2 No. 4, 1994, a copy of which is attached as Appendix 1). The definition of "management policy" has been incorporated into RFC 3198 and RFC 3060 (RFC 3198 available at http://rfc.sunsite.dk/rfc/rfc3198.html and RFC 3060 is available at http://rfc.sunsite.dk/rfc/rfc3060.html, both of which are attached hereto as Appendix 2 and Appendix 3, respectively). Using the definition that is common to both RFC 3198 and RFC 3060, policies can be defined as a set of rules to administer, manage, and control access to network resources.

Based on the above definition of management policy, replacing management policy in the claims by the above definition of management policy, claim 1 claims a method for defining a set of rules to administer, manage, and control access to network resources for controlling behavior of a management system managing at least one network element of a communication network, said method comprising:

executing a program on a processor-based device that presents a user interface for defining the set of rules to administer, manage, and control access to network resources;

receiving input from a user identifying management action to be performed by the set of rules to administer, manage, and control access to network resources; and

receiving input from a user specifying a modifiable process flow for said management policy to utilize in performing said management action

Similarly, claim 13 claims a management system managing at least one network element of a communication network, the management system comprising:

a software program stored to a data storage device, said software program executable to present a user interface for defining a set of rules to administer, manage, and control access to network resources for controlling behavior of said management system;

at least one processor-based device operable to execute said software program; and at least one input device communicatively coupled to said at least one processor-based device to allow input from a user to said software program to identify management action to be performed by the set of rules to administer, manage, and control access to network resources and to specify a modifiable process flow for said management policy to utilize in performing said management action.

Comparing Buzsaki to the claimed invention of claims 1 and 13, Buzsaki discloses a user created or modified custom error handling process executed by a process engine (col. 4, lines 1- 6, the '193 patent). Buzsaki does not disclose "a user interface for defining the set of rules to administer, manage, and control access to network resources."

Comparing Winokur et al. (the '637 patent) to the claimed invention of claims 1 and 13, Winokur et al. disclose an expert system for managing error events in a local area network. The expert system disclosed by Winokur et al. includes a knowledge base containing causal relationships between error messages and possible causes and an inference engine utilizing a causal model to capture and represent the relationship between error messages and actual causes. A user can modify and expand the knowledge base. The causal model generally consists of error messages, causes, and recommended actions. Winokur et al. do not disclose "a user interface for defining the set of rules to administer, manage, and control access to network resources."

"To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations." (MPEP 2143)

Applicants respectfully assert that Buzsaki and Winokur et al either separately or in combination do not teach or suggest all the limitations of claim 1 or claim 13.

Furthermore, applicants respectfully assert that there is no motivation to modify Buzsaki according to Winokur et al or to combine the teachings of Buzsaki and Winokur et al.

As stated above, Buzsaki discloses a user created or modified custom error handling process executed by a process engine and Winokur et al. disclose an expert system for managing error events in a local area network. In the combined invention, the user will create or modify a custom error handling process while, at the same time, the expert system would provide a recommended error handling process. Expressing this situation in layman terms, a hand operated hammer (the user provided error handling process) and a nail gun (the expert system provided error handling process) are trying to operate at once; the result of this is either an injured hammer user (the expert system taking precedence over the user provided process) or an impasse where neither process operates (similar to bus contention, both the expert system and the user provided process attempt to operate at the same time). Therefore, combining Buzsaki and Winokur et al. would render Buzsaki unsuitable for the purpose it was intended.

If the references when combined would render the prior art invention being modified unsatisfactory for its intended purpose, there is no motivation to combine the references. *McGinley v. Franklin Sports, Inc.*, 262 F.3d at 1354; *In re Gordon*, 733 F.2d at 902. Therefore, there is no motivation to integrate the teachings of Buzsaki and Winokur et al..

If the only teaching of Winokur et al. relied on by the examiner is the fact that Winokur et al. teach that the managed system is a network, that is tantamount to the assertion that one of ordinary skill in the relevant art would have been able to arrive at the applicants' invention because he/she had the necessary skills to arrive at such a conclusion. This is not an appropriate standard for obviousness. The fact that elements are known per se does not provide a motivation to combine. See *Orthokinetics Inc. v. Safety Travel Chairs Inc.*, 806 F.2d 1565, 1 USPQ2d 1081 (Fed. Cir. 1986). That which is within the capabilities of one skilled in the art is not synonymous with obviousness. *Ex parte Gerlach,* 212 USPQ 471 (Bd.App. 1980).

Thus, assuming *arguendo* that Buzsaki and Winokur et al. either separately or in combination teach or suggest all the limitations of the amended independent claims, there is no motivation to combine.

Therefore, applicants assert that a *prima facie* case of obviousness has not been established and that claims 1, 3-5, 8-16, and 19-23 are patentable over Buzsaki in view of Winokur et al.

In conclusion, in view of the above remarks, Applicants respectfully assert that the claims in this application are now in condition for allowance and respectfully request the Examiner to enter the amendments presented herein and find claims 1, 3-5, 8-16, and 19-25 allowable over the prior art and pass this case to issue.

Since the total number of claims is less than the number of claims already been paid for, no additional fees are required. However, if fees are required, they should be charged to Deposit Account No. 50-1078.

In accordance with Section 714.01 of the MPEP, the following information is presented in the event that a call may be deemed desirable by the Examiner:

JACOB N. ERLICH (617) 854-4000

Respectfully submitted,
Gary R. Klein et al., Applicants

Dated: March 30, 2005

By: [illegible]
Jacob N. Erlich
Reg. No. 24,338
Attorney for Applicant

APPENDIX 1

To be published in Journal of Network and Systems Management, Plenum Press.
Vol.2 No. 4, 1994

POLICY DRIVEN MANAGEMENT FOR DISTRIBUTED SYSTEMS

Morris Sloman

Revised 13 September 1993

Imperial College
Department of Computing
180 Queen's Gate, London SW7 2BZ
Email: m.sloman@doc.ic.ac.uk

Abstract

Separating management policy from the automated managers which interpret the policy facilitates the dynamic change of behaviour of a distributed management system. This permits it to adapt to evolutionary changes in the system being managed and to new application requirements. Changing the behaviour of automated managers can be achieved by changing the policy without have to reimplement them – this permits the reuse of the managers in different environments. It is also useful to have a clear specification of the policy applying to human managers in an enterprise.

This paper describes the work on policy which has come out of two related ESPRIT funded projects, SysMan and IDSM. Two classes of policy are elaborated – authorisation policies define what a manager is permitted to do and obligation policy define what a manager must do. Policies are specified as objects which define a relationship between subjects (managers) and targets (managed objects). Domains are used to group the objects to which a policy applies. Policy objects also have attributes specifying the action to be performed and constraints limiting the applicability of the policy. We show how a number of example policies can be modelled using these objects and briefly mention issues relating to policy hierarchy and conflicts between overlapping policies.

Keywords

Distributed systems management, network management, management policy, security policy, policy conflicts, access rules, domains.

1 Introduction

Distributed systems management[1] involves monitoring the activity of a system, making management decisions and performing control actions to modify the behaviour of the system. Policies are one aspect of information which influences the behaviour of objects within the system. **Authorisation policies** define what an manager is *permitted or not permitted* to do. They constrain the information made available to managers and the operations they are permitted to perform on managed objects (see Figure 1). **Obligation policies** define what a manager *must or must not* do and hence guide the decision making process; the manager has to interpret policies in order to achieve the overall objectives of the organisation.

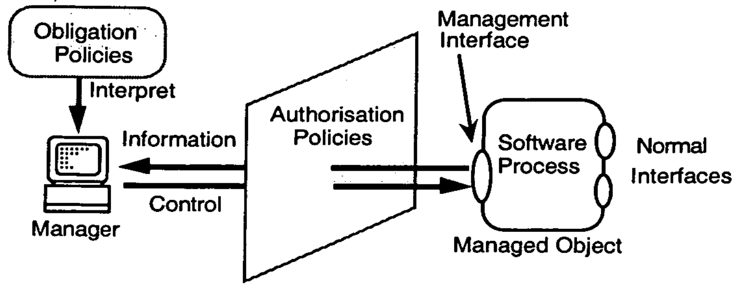

Figure 1 Policies Influence Behaviour.

Human managers are adept at interpreting both formal and informal policy specifications and, if necessary, resolving conflicts when making decisions. However the size and complexity of large distributed systems has resulted in a trend towards automating many aspects of management into distributed components. If the policies are coded into these components they become inflexible and their behaviour can only be altered by recoding. There is thus a need to specify, represent and manipulate policy information independent from management components to enable dynamic change of policies and reuse of these components with different policies.

There may be many different policies relating to the management of a large distributed system, with multiple human managers specifying policy at the same time. The complexity of the problem makes it impossible to prevent conflicts and inconsistencies, so the policy service must support analysis, wherever possible, to detect these and at least warn human users of potential conflicts and inconsistencies.

This paper presents the common policy concepts being used by two Esprit funded Projects – SysMan and IDSM which are implementing distributed management applications based on the use of domain and policy services.

[1] We consider a communications network to be a distributed subsystem providing a communications service, within an overall distributed system which may include various other services such as storage, directory, time etc. Both services and applications running above them have to managed. The concepts described in this paper can be applied to management of networks, telecommunication systems or any other distributed systems.

2 Management Framework

In this section we explain the concept of domains which are important for grouping objects to which policies apply and show how management applications, domain and policy services fit within an overall management architecture.

2.1 Domains

Management of a distributed information system cannot be centralised in a single human or automated entity but must be distributed to reflect the distribution of the system being managed. Management must thus be structured to partition and demarcate responsibility amongst the multiple managers. This structuring could reflect physical network connectivity, structuring of the distributed application or possibly reflect the hierarchical management structure (for example corporate headquarters, regional, site, departmental, and section management) found in many organisations. There will be a variety of managers fulfilling different functions and operating in different contexts, but having responsibilities for the same object. For example the maintenance engineer and the user of a workstation have different management responsibilities for the same workstation. The management structure must be able to model these overlapping responsibilities. Domains provide the framework for partitioning management responsibility by grouping objects in order to specify a management policy or for whatever reason a manager wishes.

A **management domain** is a collection of managed objects which have been explicitly grouped together for the purposes of management. More concretely, a domain is a managed object which maintains a list of references to its member managed objects. If a domain holds a reference to an object, the object is said to be a **direct member** of that domain and the domain is said to be its **parent**.

Since a domain is itself a managed object, it may be a member of another domain and is said to be a **subdomain** of its parent. Subdomains are the means of flexibly partitioning a large group of objects and applying different policies to different subgroups or assigning responsibility for applying policy to different managers. Members of a subdomain are **indirect** members of the parent domain. Managed objects can be direct or indirect members of multiple domains. When an object is a direct member of multiple domains, the parent domains are said to **overlap**. Overlapping domains thus have one or more member objects in common. Domains are similar to the notion of a directory commonly found in hierarchical file systems. More information on domain concepts can be found in [1,2,3] and a detailed specification of the domain service in [4].

2.2 Management Architecture

Figure 2 shows the overall distributed management system architecture. Rather than implement a single monolithic management application (MA) to perform all aspects of management, we have an extensible set of management applications which have a consistent user interface. These make use of a common set of underlying management services for monitoring and manipulating domains and policies. The management objects may interact using various communication services to meet the requirements of particular applications.

Each MA may have its own managed objects grouped into domains and may have one or more human managers depending on the scale of the application and the need for partitioning responsibility. A manager "sees" all the objects (within domains) for which he is responsible or can access. This is analogous to accessing files and devices in a Unix system via the hierarchical Unix directory. A MA will have a user interface (UI) specific to that application but the UI, dealing with browsing the domain hierarchy, and specifying policy will have a common "look and feel" across applications.

Although Figure 2 shows a layered hierarchy, this should not be taken too literally. For example there is both a management application part and a service part for configuration, security and monitoring. The management applications are themselves distributed and may directly access distributed processing or communication services without using intermediate layers. The common management services and the distributed processing services may also be implemented by distributed components. The communication system and the distributed processing services should themselves be managed by the management applications they support. All management applications interpret and apply policies and are subject to security to control access. Further information on the Management Architecture can be found in [5] and how it is being implemented in the IDSM project in [6].

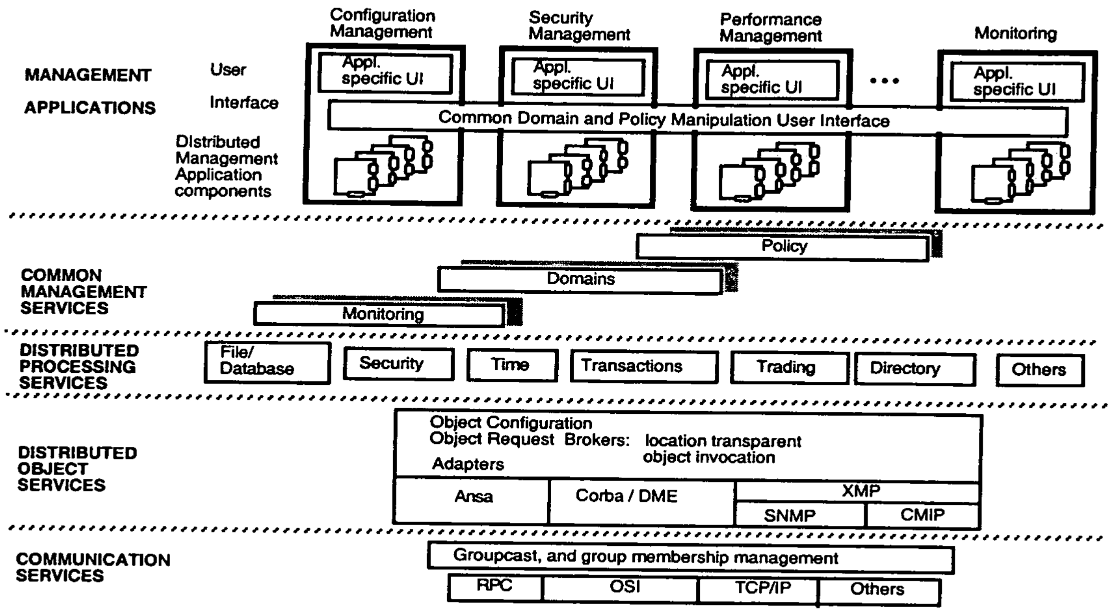

Figure 2 Distributed management system architecture

3 Management Policy

In this section we elaborate on the concepts of policy introduced in section 1 and show how domains can be used to specify the scope of a policy.

3.1 Policy Classification

In an object oriented approach, the external behaviour of an object defines how it interacts with other objects in its environment. We refine the concept of policy to be the information which influences the interactions between a **subject** and a **target** and so the policy specifies a relationship between the subject and target. Multiple policies may apply to any object as it may be the subject or target of many policies.

3.1.1 Authorisation Policy

Authorisation policy defines what activities a subject is permitted to do in terms of the operations it is authorised to perform on a target object. In general an authorisation policy may be positive (permitting) or negative (prohibiting) i.e. not permitted = prohibited. Authorisation policies are considered *target based* in that there is a reference monitor associated with the target which enforces the policy and decides whether an activity is permitted or prohibited. We do not consider mandatory military type policies in this paper.

Activity based authorisation

The simplest policies are expressed purely in terms of subject, target and activity:

* John is permitted to read file F1 (Positive)
* John is prohibited to read, write or execute file F3 (Negative)

A target based reference monitor can then make a decision based on the subject and operation although an implicit subject may be specified.

* Any object is permitted to read file F1 (Positive)
* Any object is prohibited to write file F3 (Negative)

State Based authorisation

State based authorisation policies include a predicate based on object state (i.e. a value of an object attribute) in the policy specification. These are common in database access control and safety critical systems:

* John is permitted to read personnel records where employment grade <10
* The operator is prohibited from performing close_valve on reactor when reactor.Temp > 100
* Managers with current location = planning office are permitted to read expansion plans (i.e. they are prohibited when visiting other locations – this assumes current location is an attribute of a manager).

3.1.2 Obligation Policies

Obligation policy defines what activities a subject must (or must not) do. The underlying assumption is that all subjects are well behaved, and attempt to carry out obligation policies with no freedom of choice. This may be true of automated subjects but will in general not be true of human subjects. Obligation policies are *subject based* in that the subject is responsible for interpreting the policy and performing the activity specified.

Activity Based Obligations

Simple obligation policies can also be expressed in terms of subject, target and activity, but may also specify an event which triggers the activity.

* The company director must protect the assets of company XYZ (Positive)
* On error count > 100 monitoring agent must send warning message to operator (Positive, event triggered)
* The standby manager must not perform any control actions (Negative)
* Employees must not talk about their jobs to the press (Negative)

State Based Obligation

An obligation may also be specified in terms of a predicate on object state. In some cases this can be used to select subject or target objects to which an obligation policy applies.

* Controller must control boiler temperature such that 50 < boiler.temp > 100
  (Positive obligation in terms of target state)
* Managers must perform reset on links with error count > 50
  (Positive obligation on selected targets based on state)
* Managers with version < 1.5 must not perform diagnostic test A500
  (Negative obligation applying to selected subjects)

3.1.3 Discussion

Authorisation policies are specified to protect target objects and are usually implemented using security mechanisms in the operating system as subjects cannot be trusted to enforce them. Obligation policies are implemented by the management system i.e. interpreted by managers which must be trusted. Authorisation policies are less dynamic than obligation policies. For example obligation policies may be triggered by an event which results in an action being performed but are effectively dormant until the event occurs again. We have not identified any need for event based authorisation policies.

A negative obligation may appear to be the same as a negative authorisation but the responsibility for preventing the activity lies with the subject rather than with a target based reference monitor. This assumes the subject is well behaved and trusted. The subject may in fact be authorised to perform the activity and the negative obligation is activated to stop it on a temporary basis. For example, a standby manager may normally be authorised to perform control actions but a negative obligation stops the standby manager. Although it would be feasible to transform a negative obligation into a negative authorisation, this may be inconvenient due to the controls and overheads involved in introducing authorisation policies into the system. Transformation to an authorisation policy is necessary if the subject cannot be trusted to perform a negative obligation.

State based policies are more difficult to implement than activity based policies. A reference monitor would have to query subject or target objects to check their state in order to determine whether to permit an action for authorisation policies [7]. A state based obligation policy e.g. to maintain boiler temperature between 50 and 100 cannot be directly interpreted by an automated manager as it needs "intelligence" to work out how to achieve the required goal. The obligation could be refined into the following activity based obligations (which also allow for some time lag in the boiler heater affecting temperature).

* On boiler.temp < 52 controller must switch on boiler heater
* On temperature > 98 controller must switch off boiler heater

Negative state based policies can sometimes be transformed into positive ones by modifying the predicate. For example

* The operator is permitted to close valve on reactor when reactor.Temp ≤ 100

is equivalent to the negative policy defined in 3.1.1 above. This assumes there is an implicit negative authorisation policy forbidding any access unless a positive authorisation policy permits it. Combining positive and negative policies can result in conflicts [8].

Wies [9] has a similar policy mode classification to the above but extends this with additional classification criteria such as lifetime, geographical scope, organisational structure, type of service, type or functionality of targets, management functionality to which the policy applies. These criteria are then used to derive attributes for a policy template.

3.2 Policy Constraints

A constraint can optionally be defined as part of a policy specification to restrict the applicability of the policy. It is defined as a predicate referring to global attributes such as time or action parameters, as explained below.

Temporal Constraints specify time limits before, after or between which a policy applies e.g. between 09.00 and 17.00 or before 31 December 1994. They may be used to specify a validity time or expiry time for a policy.

Parameter value constraints define permitted values for management operations. For example the security policy that passwords must be greater than 6 characters in length and contain at least one non-alphabetic character can be considered a constraint on a change password operation parameter.

Preconditions could define the resources which must be available for a management policy to be accomplished. For example, a dynamic load balancing policy could specify that processes may be migrated to a machine in domain D1 with load < 60% up to limit of 4 processes per machine. Budget allocation is often considered a policy decision.

Other constraints which limit the applicability of a policy can be defined as part of a selection expression for a state based policy based on object attributes to select the set of subject or target objects within a domain to which the policy will be applied (see section 3.4 below).

3.3 Policy as Relationship Objects

Policies encapsulate a representation of information affecting component behaviour so we treat them as objects which provide operations for querying or changing policies [10]. A policy service then provides the operations for creating, deleting, storing and retrieving policy objects. Policy scope is specified using domains, so the policy service must also provide the ability to identify what policies apply to a domain and then use the domain service to identify the objects within the domain. There are advantages in treating policies as managed objects and structuring them into domains, so that an authorisation policy can be defined to control which managers are permitted to modify a set of policies or to define "meta policies" about policies (see section 4.4.).

Another reason for an object oriented approach to policy specification is that it is useful to be able to define a policy class which defines most of the attributes of a particular policy. When specifying policies for a particular application, multiple instances of that policy can then be created, with remaining policy attributes being defined for the particular instance. The policy class is like a template with values for specific attributes being provided when a policy instance is created.

Policies are not active objects in that they do not instigate management operations. Managers are the active objects which are responsible for interpreting an obligation policy and performing the activities specified. A reference monitor uses information such as an access control list derived from authorisation policies to decide whether to permit an operation [11].

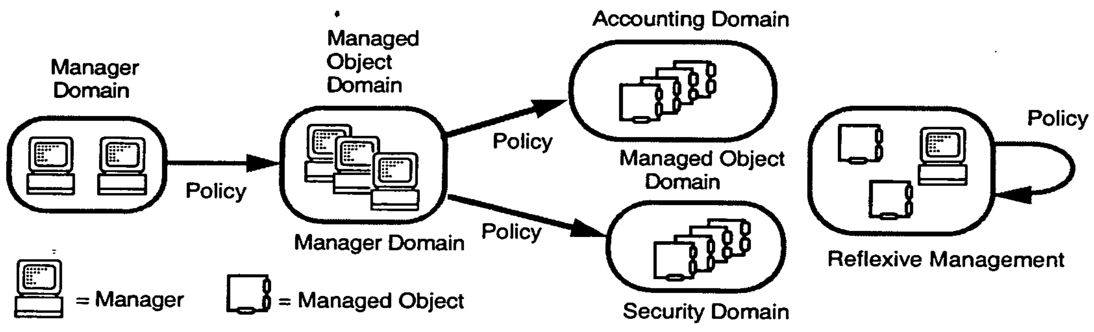

Figure 3 Typical Management Relationships

Figure 3 also shows that a policy may specify a reflexive relationship, whereby managers are members of the managed domain and so could be both subjects and targets of the management policy. This reflects the fact that managers may manage themselves in some circumstances e.g. authorised to approve their own expenses. There is no restriction on the type of object within a single domain so the policy may need to specify the type of object to which it applies unless it is applicable to all objects in the domain.

The OSI Manager, agent, managed object relationships [12] can be modelled by 2 sets of policies – policies which specify the relationship between the Manager and the agent and those which specify the relationship between the agent and the managed objects for which it is responsible. The latter are probably implicit as OSI management does not really consider agents to be intelligent and make management decisions.

3.4 Policy Scope Specification

In very large systems, the number of objects is so large that it is impractical to specify policies for individual objects. Instead it should be specified for sets of objects. The set of objects to which a policy applies could be specified in terms of object attributes e.g. a particular type of object or those objects in a particular state. A search over all reachable objects, within a distributed system, to determine these sets is impractical. The number of reachable objects within a large scale distributed system, is potentially millions and is not known a priori. The selection of objects is thus limited to be within the **scope** of a domain. This limits the search space for object selection to a predefined set to make sure the selection terminates within a defined time. For example the policy

* Kevin must install new kernel on workstations with workstation type=Sparc IPX

is impractical as there are potentially millions of such workstations connected to the internet, so the scope should be limited as in the following policy.

* Kevin must install new kernel on workstations in domain dse.doc.ic.ac.uk with workstation type=Sparc IPX

Another advantage of specifying policy scope in terms of domains is that objects can be added and removed from domains to which policies apply without having to change the policies.

3.4.1 Propagation to Subdomains

Policies apply to sets of objects within domains, but domains may contain subdomains. To avoid having to respecify policy for each subdomain, policy applying to a parent domain, should **propagate** to member subdomains of the parent. For example a policy applying to an organisation should also apply to departments within that organisation. A subdomain is said to **inherit**, the policy applying to parent domains (but this is not the same as object oriented inheritance). With policy propagation, a policy specified for a domain is applied recursively to all direct and indirect members of that domain. For example, in Figure 4 the policy specified between D1 and D2 will propagate to managers in D2 and the managed objects in D4 and D5.

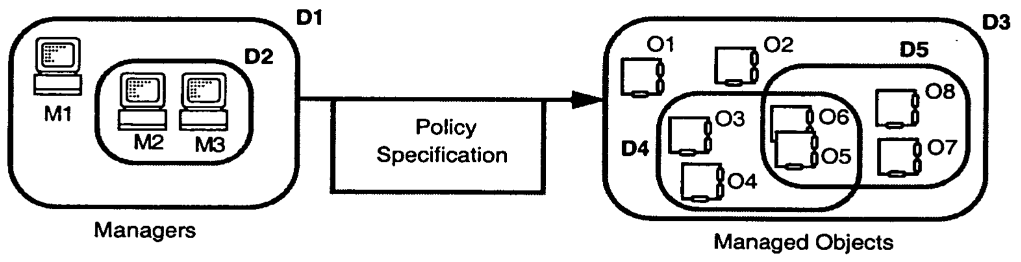

Figure 4 Policy Propagation

It should be possible to override the default policy propagation either at the policy or domain level. A policy may specify that it applies only to direct members of the target or subject domains. A domain attribute may specify that any policy applying to the domain does not propagate to indirect members, irrespective of what is specified in the policy.

In order to efficiently determine the policies applying to a domain and hence to an object within it, the domain must hold references to those policies.

3.4.2 Set Selection

A policy should be able to select the set of subject or target objects within a domain to which it applies using a predicate based on the values of object attributes. The simplest case occurs when a policy applies to all objects in a domain. The set of objects to which a policy applies has to be evaluated at the time the policy is interpreted because domain membership can change dynamically. Object selection is based on *Scope Expressions* which define a (possibly empty) set of objects and are based on combinations of the following:

i) The object itself.

ii) Direct and indirect members of the domain i.e. policy is applied recursively to all subdomains which are members or indirect members of the domain.

iii) Limited propagation. i.e. policy is applied recursively to a limited number of levels of subdomains which are members the domain.

iv) A predicate based on object attributes is used to select objects. For example the policy applies to objects of a particular type or in a particular state. A *location constraint* may limit the applicability of an authorisation policy in terms of the location from which operations on the objects can be invoked e.g. a file can only be read from terminals in a particular office. Evaluating a set of objects, using a predicate based on an object attribute value together with policy propagation, could be very expensive to implement in a distributed system.

v) A set expression in terms of members of the domain can be evaluated to give a set.

vi) Any objects - this allows an implicit scope. For example any manager may be authorised to perform an operation on an object. "Any" is only permitted as the subject scope for an authorisation policy or the target scope for an obligation policy.

The use of "any" as the subject of an obligation policy does not make sense as no specific subject has the responsibility to carry out the actions. We have not identified a use for "any" as the target for authorisation policy.

3.4.3 Scope Expressions

Scope Expressions always return a set of objects and are defined as follows:-

```
SE ::=          "ANY" / SC_EXPR

SC_EXPR ::= object |
            { object } |
            *object |
            * NUMBER object |
            SC_EXPR + SC_EXPR |
            SC_EXPR - SC_EXPR |
            SC_EXPR ^ SC_EXPR |
            select( pred, SC_EXPR ) |
            ( SC_EXPR )
```

Operators

+	set union
-	set difference
^	set intersection
*	this returns a set that contains all direct and indirect members of the domain if it is applied on a domain object ; otherwise it returns a set that contains the object itself.

* NUMBER a set that contains all direct and indirect members of the domain as far down as the NUMBER'th level if it is applied on a domain object returns. That is, *1D1 gives the set of direct members of domain D1. If applied to an object it returns a set that contains the object itself.

{ } returns a set that contains the object on which it is applied i.e. it converts a single object to a set containing that object. It is only needed for set theory consistency, as the operators (+, -, ^) are only applied to sets of objects. There is no ambiguity if it is omitted.

object shorthand version or saying { object }

select(*pred, sc_expr*) returns a sub-set of the set returned by *sc_expr* with the members of the selected set determined by the predicate. The predicate will typically be a function which is applied to all members of the set returned by *sc_expr*.

The interpretation of the expressions is from left to right.

Operators can be divided into two categories. The first category includes the set operators (+, -, ^) which are applied to sets of objects. The second one includes the object operators (*, { }) that are applied to objects and return a set of objects. The set that the object operators return is evaluated by traversing the domain hierarchy starting with the domains referenced in the expression.

For example, referring to Figure 4:

- `*1D4-O3+*1D5` = union of direct members of D4 (except O3) and D5 = `{O4,O5,O6,O7,O8}`
- `*D4^*D5` = intersection of direct and indirect members of D4 and of direct and indirect members of D5 = `{O6, O5}`.
- `*D3` = all direct and indirect members of D3 = `{O1,O2,O3,O4,O5,O6,O7, O8,D4,D5}`
- `select(type!=Domain, *1D4-O3+*1D5)` = non-domain members of the set of the union of direct members of D4 (except O3) and D5 = `{O4,O5,O6,O7,O8}`.

4 Example Policy Objects

A policy object specification defines the following attributes:

i) Modality: positive or negative authorisation, positive or negative obligation (i.e. A+, A-, O+, O-)
ii) A subject which defines one or more manager domain scopes
iii) A target which defines one or more managed domain scopes affected by the activity
iv) An activity which define a set of actions or permitted operations
v) Constraints which apply to the activity

Example policies with these characteristics are given below.

4.1 Access Rules

An access rule is a simple example of a management authorisation policy which specifies a relationship between managers (in a subject scope domain) and managed objects (in a target scope domain) in terms of the management operations permitted on objects of a specific type.

The access rule may also define constraints on these operations (see 3.2 above) and make use of scope expressions to select subsets of the objects within the subject or target domains. In Figure 5, operations OpA and OpB are permitted on objects of type T1 and operations OpX and OpZ on objects of type T2. These operations can only be performed between hours of 09.00 to 17.00. Examples of the use of access rules for service specification can be found in [2].

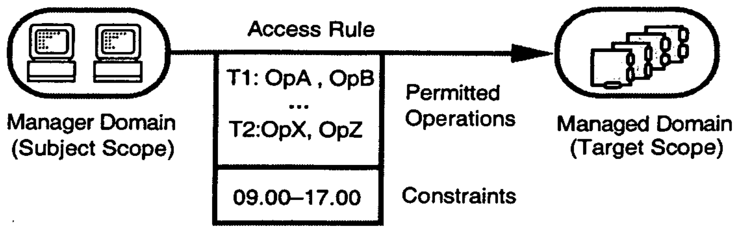

Figure 5 Access Rules

4.2 Domain Membership Policy

A manager can specify the initial membership of a domain by specifying an object selection predicate for searching a database or another domain, but this is not provided as part of the basic domain service. Membership policies are then needed to constrains the objects which can be subsequently created in the domain or included from another domain. Other membership policies relate to the number of objects permitted in a domain. Example membership policies are given below.

- Only objects which implement a particular interface type can be members of the domain i.e. any subject is permitted to include or create objects of type T in target domain Dt.

  ```
  A+ any {include X, create X} Dt when X.type=T
  ```

- A domain of managers can have only a single manager to prevent conflicts between multiple managers.

  ```
  A+ any {include, create} Dt when Dt.membernum = 0
  ```

- There must always be at least two objects in a domain for backup purposes. This also requires an obligation policy for a manager in domain Ds, on receiving a failure event, to delete the failed object and create a new one.

  ```
  A- any {remove, delete} Dt when Dt.membernum > 2;
  O+ on fail(X) Ds {delete X; create X} Dt
  ```

- An object should always be a member of at least one domain if it is to be managed using the domain based management applications mentioned in section 2, so a policy could specify that removing it from a target domain Dt is only permitted if has more than one parent.

  ```
  A+ any {remove X} Dt when X.parentnum > 1
  ```

4.3 Delegation

In some applications a manager may delegate an activity to a proxy manager (or agent) to perform on his behalf. There is a need to control to whom the managers can delegate and what operations they can delegate. This type of policy requires two subject domains – for the delegator and delegatee. The policy shown in Figure 6 permits Managers in Domain D1 to delegate the right to perform operations OpA and OpC on target objects of Type T1 in Domain D2, to a proxy manager in Domain D3. The policy expires after 31 December 1994.

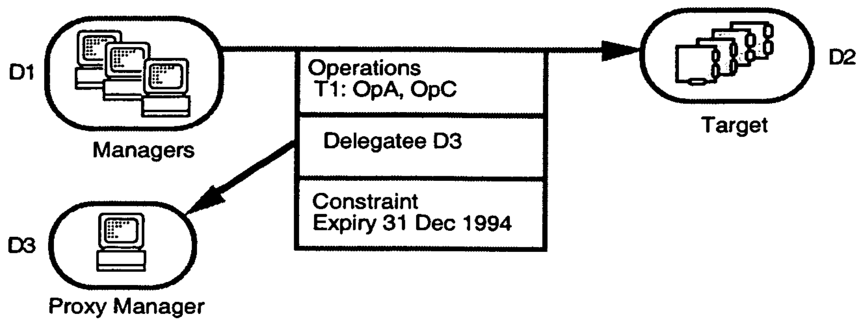

Figure 6 Delegation of rights.

The implementation of this type of policy requires extended access control lists which contain information on delegatees as well as subjects.

4.4 Security Administrator

A security administrator (SA) in a commercial environment would typically create access rules (ARs) for other subjects (excluding himself) to access specific target resources. The authorisation policy applying to the SA should limit the subjects for whom he can create ARs, the target objects to which the rules can apply and the operations specified by the ARs. This is a meta-policy about managing policy objects (access rules) and also specifies a relationship between multiple domains. It is the most complicated of the examples. Figure 7 shows there are a number of scopes which limit the ARs which can be created. *AR permitted subjects* specifies to whom access can be given, the *AR permitted targets* define the set of target objects to which access can be given, and an *AR permitted operations* define the set of operations which can be included in access rules. The principle of separation of responsibility means the SA should not be permitted to give access to himself, so the subject should not be a member of the *AR permitted subjects*. If we consider ARs as objects which are created in domains, there is a target domain in which the SA is permitted to create ARs.

Interpreting policy objects such as those relating to a security administrator is considerably more complicated than simple access rules. It would be the function of the policy service to interpret and enforce this policy as it relates to creating policy objects.

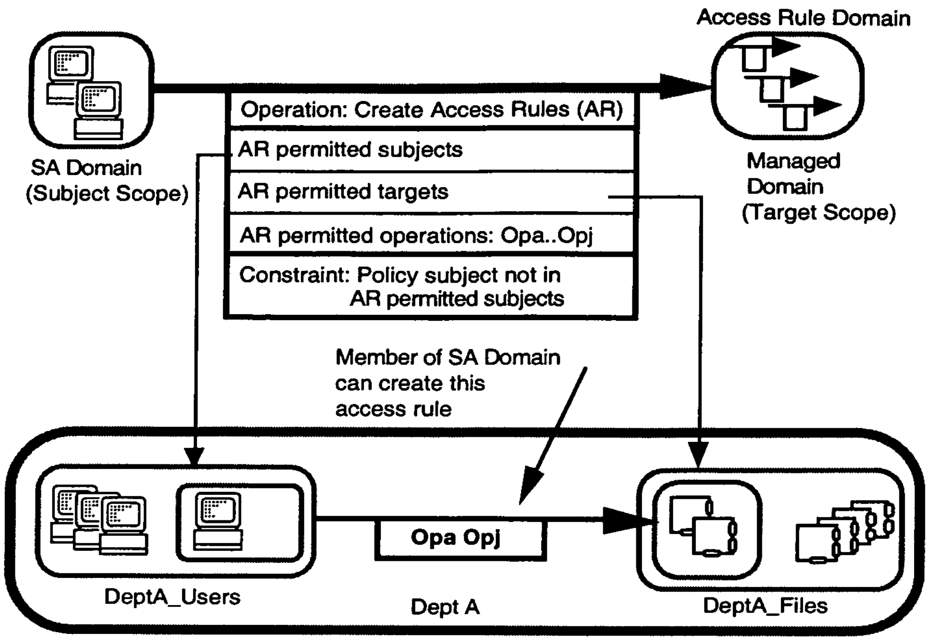
Access Rule Domain
SA Domain
(Subject Scope)
Operation: Create Access Rules (AR)
AR permitted subjects
AR permitted targets
AR permitted operations: Opa..Opj
Constraint: Policy subject not in AR permitted subjects
Managed Domain
(Target Scope)
Member of SA Domain can create this access rule
Opa Opj
DeptA_Users
Dept A
DeptA_Files

Figure 7 Authorisation Policy for a Security Administrator (SA)

4.5 Responsibility

The concept of responsibility can be modelled as an obligation policy. For example consider that manager X has responsibility to update software in a domain of workstations. Manager Y is his superior and has ultimate responsibility to determine the work is carried out correctly. This is modelled using two different obligation policies, one to perform the updating and one to indicate the responsibility relationship as shown in Figure 8.

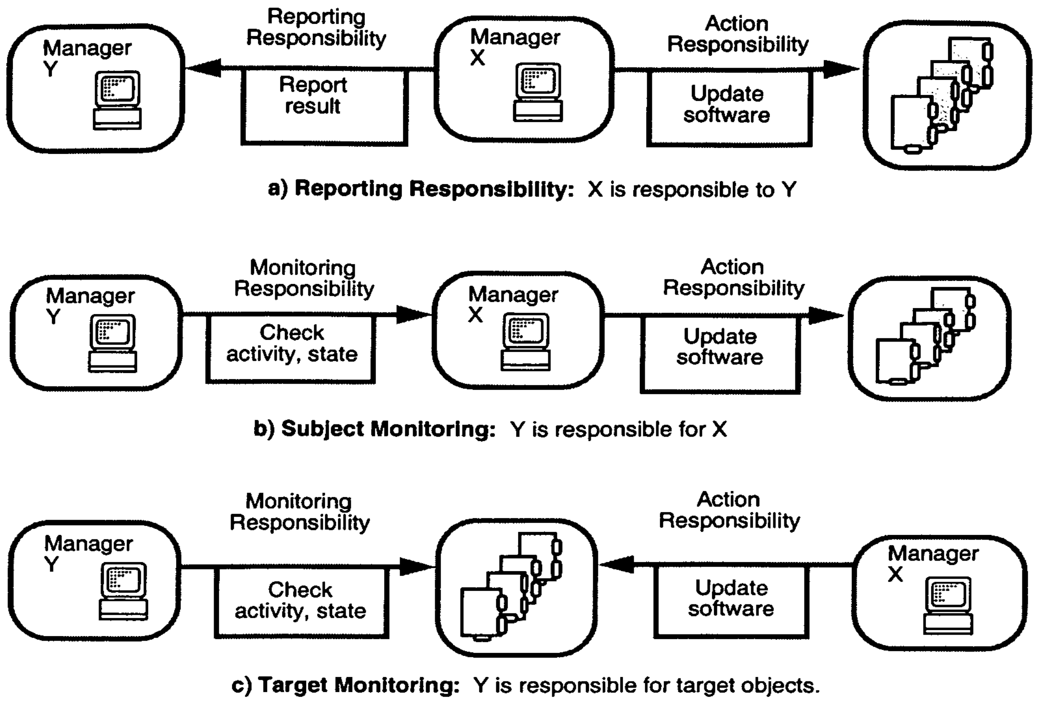

Figure 8 Modelling Responsibility as Obligation Policies.

5 Policy Implementation Issues

A **Policy Dissemination Function** transforms policies into a form suitable for interpretation or enforcement and sends obligation policies to managers in the subject domain and authorisation policies to reference monitors associated with objects in the target domain. An example transformation is an authorisation policy object into an access control list (ACL) entry or capability. ACLs are stored with the target domain and have to be propagated to nested subdomains [13]. The authorisation policies have been applied to specifying service access rights for cellular networks [2].

The generalised policy concepts were derived from our initial work on access control policy, so the authorisation policy aspects are further advanced than the obligation policy concepts. We are experimenting with a notation which can be used for both authorisation and obligation policies as they have similar attributes but the respective implementation mechanisms are very different. Obligation policies are of the form

O+ | O- [on <event>] <subject> {actions} <target> [when <constraint>]

where the action is triggered by an event occurring and the action could be specified by a C language procedure. The constraint is a predicate which is evaluated when the event occurs and can be used to inhibit the action being performed. The event and constraint expression are optional. In particular this has been applied to a monitoring service where these policy rules can be used to combine and filter events or generate higher level event reports [14]. We are developing graphical tools for specifying both authorisation and management policy.

The IDSM partners are using their proprietary Network Management platforms to implement policies and domains as OSI managed objects in special Management Information Bases (MIBs) [6, 15] supported by service managed objects. These implementations use the

Network Management Option within OSF's DME to access OSI and SNMP managed objects via the XMP interface [16]. A management agent uses the authorisation policies to make access control decisions for management operations. Reporting obligation policies for generating event messages are being translated into event forwarding discriminator managed objects. The SysMan project is using ANSAware [17], which is a distributed object oriented programming environment so domains, policies and the distributed servers which store them can be directly implemented as Ansa objects. One of the commercial partners is porting this implementation to a CORBA platform [18].

6 Manager Roles

6.1 Conceptual Issues

The concept of a role is well understood in enterprise modelling and there is an extensive literature relating to role theory [19]. Role Theory postulates that individuals occupy positions in an organisation. Associated with each position is a set of activities including required interactions that constitute the role of that position.

A **manager role** is defined as the set of authorisation and obligation policies for which a particular manager position is the subject. A role thus identifies the authority, responsibility, functions and interactions, associated with a position within an organisation. Example manager positions could include managing director, security administrator, operations manager, operator responsible for North Region. A person may be assigned to one or more roles and multiple individuals can share a single role.

A **manager position** defines a particular position within an organisation, such as financial manager, managing director, to which different people may be assigned over a period of time. The role refers to a manager position rather than a particular person, because people are frequently assigned to new roles, and it would be very time consuming to change policies which reference that person. Within the management environment the functions and authority of a human or automated manager is defined in terms of the obligation and authorisation policies which apply to the manager position. This defines the overall functionality of the position. A role may also relate to an automated manager, although it is less likely to be frequently reassigned to new roles.

Policies which have propagated down to a position, but do not explicitly reference the position are not included in the set of role policies. Propagated policies are part of the organisation policies as they may also apply to different roles or objects, and are not specific to the manager position which is the subject of the role.

It would be very useful to be able to parameterise a role with specific positions and target domains. It would thus be possible to define the role policy set as a class from which particular instances can be created. For example a role could be defined for a region manager and this could be used to create North, South, East and West region manager roles. Each of the 4 roles instances relates to different manager positions each with their own specific target domains, but specifies the same policy activities and constraints for each manager position.

6.2 Implementation Issues

It is assumed that the humans occupying roles will perform management functions related to the distributed system and so have to be represented within the system by an adapter object which interacts with a suitable presentation device (workstation or terminal). We now show how the domains described in section 2.1 can be used to represent Users and Positions.

The policies applying to a person are defined in terms of a **User Representation Domain (URD)** which is a persistent representation of the person or human manager. When the person logs into the system an adapter object is created within the URD to interact with the person's

APPENDIX 2

Network Working Group
Request for Comments: 3444
Category: Informational

A. Pras
University of Twente
J. Schoenwaelder
University of Osnabrueck
January 2003

On the Difference between Information Models and Data Models

Status of this Memo

Copyright Notice

Abstract

There has been ongoing confusion about the differences between Information Models and Data Models for defining managed objects in network management. This document explains the differences between these terms by analyzing how existing network management model specifications (from the IETF and other bodies such as the International Telecommunication Union (ITU) or the Distributed Management Task Force (DMTF)) fit into the universe of Information Models and Data Models.

This memo documents the main results of the 8th workshop of the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF) hosted by the University of Texas at Austin.

Table of Contents

1. Introduction 2
2. Overview 2
3. Information Models 3
4. Data Models 4
5. Security Considerations 6
6. Acknowledgments 6
7. Normative References 6
8. Informative References 7
9. Authors' Addresses 7
10. Full Copyright Statement 8

. Introduction

Currently multiple languages exist to define managed objects. Examples of such languages are the Structure of Management Information (SMI) [1], the Structure of Policy Provisioning Information (SPPI) [2] and, within the DMTF, the Managed Object Format (MOF) [3]. Despite the fact that multiple languages exist, a number of people still believe that none of these languages really suits all needs.

There have been many discussions to understand the advantages and disadvantages, as well as the main differences, between various languages. For instance, the IETF organized a BoF on "Network Information Modeling" (NIM) at its 48th meeting (Pittsburgh, August 2000). During these discussions, it turned out that people had a different understanding of the main terms, which caused confusion and long arguments. In particular, the meaning of the terms "Information Model" (IM) and "Data Model" (DM) turned out to be controversial.

In an attempt to address this issue, the IRTF Network Management Research Group (NMRG) dedicated its 8th workshop (Austin, December 2000) to harmonizing the terminology used in information and data modeling. Attendees included experts from the IETF, DMTF and ITU, as well as academics who do research in this field (see the Acknowledgments section for a list of participants). The main outcome of this successful workshop -- a better understanding of the terms "Information Model" and "Data Model" -- is presented in this document.

Short definitions of these terms can also be found elsewhere (e.g., in RFC 3198 [8]). Compared to most other documents, this one explains the rationale behind the proposed definitions and provides examples.

2. Overview

One of the key observations made at the NMRG workshop was that IMs and DMs are different because they serve different purposes.

The main purpose of an IM is to model managed objects at a conceptual level, independent of any specific implementations or protocols used to transport the data. The degree of specificity (or detail) of the abstractions defined in the IM depends on the modeling needs of its designers. In order to make the overall design as clear as possible, an IM should hide all protocol and implementation details. Another important characteristic of an IM is that it defines relationships between managed objects.

DMs, conversely, are defined at a lower level of abstraction and include many details. They are intended for implementors and include protocol-specific constructs.

```
            IM                --> conceptual/abstract model
             |                    for designers and operators
+----------+---------+
|          |         |
DM         DM        DM       --> concrete/detailed model
```

for implementors

The relationship between an IM and DM is shown in the Figure above. Since conceptual models can be implemented in different ways, multiple DMs can be derived from a single IM.

Although IMs and DMs serve different purposes, it is not always possible to precisely define what kind of details should be expressed in an IM and which ones belong in a DM. There is a gray area where IMs and DMs overlap -- just like there are gray areas between the models produced during the analysis, high-level design and low-level design phases in object-oriented software engineering. In some cases, it is very difficult to determine whether an abstraction belongs to an IM or a DM.

3. Information Models

IMs are primarily useful for designers to describe the managed environment, for operators to understand the modeled objects, and for implementors as a guide to the functionality that must be described and coded in the DMs. The terms "conceptual models" and "abstract models", which are often used in the literature, relate to IMs. IMs can be implemented in different ways and mapped on different protocols. They are protocol neutral.

An important characteristic of IMs is that they can (and generally should) specify relationships between objects. Organizations may use the contents of an IM to delimit the functionality that can be included in a DM.

IMs can be defined in an informal way, using natural languages such as English. An example of such an IM is provided by RFC 3290 [9], which describes a conceptual model of a Diffserv Router and specifies the relationships between the components of such a router that need to be managed. Within the IETF, however, it is exceptional that an IM be explicitly described, and even more that the IM and DM be specified in separate RFCs. In such cases, the document specifying the IM is usually an Informational RFC whereas the document defining the DM usually follows the Standards Track [4]. In general, most of

the RFCs that define an SNMP Management Information Base (MIB) module also include some kind of informal description explaining parts of the model behind that MIB module. Such a model can be considered as a document of an IM. An example of this is RFC 2863, which defines "The Interfaces Group MIB" [10]. But most MIB modules published to date include only a rudimentary and incomplete description of the underlying IM.

Alternatively, IMs can be defined using a formal language or a semi-formal structured language. One of the possibilities to formally specify IMs is to use class diagrams of the Unified Modeling Language (UML). Although such diagrams are still rarely used within the IETF, several other organizations routinely use them for defining IMs, including the DMTF, the ITU-T SG 4, 3GPP SA5, the TeleManagement Forum, and the ATM Forum. An important advantage of UML class diagrams is that they represent objects and the relationships between them in a standard graphical way. Because of this graphical representation, designers and operators may find it easier to understand the underlying management model. Although there are other techniques to graphically represent objects and relationships (e.g., Entity-Relationship (ER) diagrams), UML presents the advantage of being widely adopted in the industry and taught in universities. Also, many tools for editing UML diagrams are now available. UML is standardized by the Object Management Group (OMG) [5].

In general, it seems advisable to use object-oriented techniques to describe an IM. In particular, the notions of abstraction and encapsulation, as well as the possibility that object definitions include methods, are considered to be important.

4. Data Models

Compared to IMs, DMs define managed objects at a lower level of abstraction. They include implementation- and protocol-specific details, e.g. rules that explain how to map managed objects onto lower-level protocol constructs.

Most of the management models standardized to date are DMs. Examples include:

o Management Information Base (MIB) modules defined within the IETF. The language (syntax) used to define these DMs is called the "Structure of Management Information" (SMI) [1] and is derived from ASN.1 [6].

o Policy Information Base (PIB) modules, developed within the IETF. Their syntax is defined by the "Structure of Policy Provisioning Information" (SPPI) [2], which is close to SMI and is also derived from ASN.1 [6].

o Management Information Base (MIB) modules, originally defined by the ISO and currently maintained and enhanced by the ITU-T. The syntax of these DMs is specified in the "Guidelines for the Definition of Managed Objects" (GDMO) [7]. GDMO MIB modules make use of object-oriented principles.

o CIM Schemas, developed within the DMTF. The DMTF publishes them in two forms: graphical and textual. The graphical forms use UML diagrams and are not normative (because not all details can be represented graphically). They can be downloaded from the DMTF Web site in PDF and Visio formats. (Visio is a tool to draw UML class diagrams.) The textual forms are normative and written in a language called the "Managed Object Format" (MOF) [3]. CIM Schemas are object-oriented.

Because CIM Schemas support a graphical notation whereas IETF MIB modules do not, designers and operators may find it easier to understand CIM Schemas than IETF MIB modules. One could therefore argue that CIM Schemas are closer to IMs than IETF MIB modules.

The Figure below summarizes these examples. The languages that are used to define the DMs are shown between brackets.

```
                        IM                                   --> IM
                         |
      +----------+-------+-------+-------------+
      |          |               |             |
     MIB        PIB          CIM schema     OSI-MIB         --> DM
    (SMI)      (SPPI)          (MOF)        (GDMO)
```

To illustrate what details are included in a DM, let us consider the example of IETF MIB modules. As opposed to IMs, IETF MIB modules include details such as OID assignments and indexing structures. The relationships defined in the IM are implemented as OID pointers or realized through indexing relationships specified in INDEX clauses. Many other implementation-specific details are included, such as MAX-ACCESS and STATUS clauses and conformance statements.

A special kind of DM language is the SMIng language defined by the NMRG. This language was designed at a higher conceptual level than SMIv1/SMIv2 and SPPI. In fact, one of the intentions behind SMIng was to stop the proliferation of different DM languages within the IETF and to harmonize the various models. As a result, MIB and PIB

modules defined in SMIng can be mapped on different underlying protocols. There is a mapping on SNMP and another mapping on COPS-PR. SMIng is therefore more protocol neutral than other IETF approaches. It also supports some object-oriented principles and provides extension mechanisms that allow the addition of new features (e.g., the support for methods). New features can then be used when they are supported by the underlying protocols, without breaking SMIng implementations. Still, SMIng should be considered a DM. For instance, to express relationships between managed objects, techniques such as UML and ER diagrams still give better results because these diagrams are easier to understand.

Note that the IETF SMING Working Group took a different approach and decided not to use the SMIng language defined by the NMRG. Instead, the SMING Working Group is developing a third version of SMI (SMIv3) that is primarily targeted towards SNMP, and which incorporates only some of the ideas developed within the NMRG.

5. Security Considerations

The meaning of the terms Information Model and Data Model has no direct security impact on the Internet.

6. Acknowledgments

The authors would like to thank everyone who participated in the 8th NMRG workshop (in alphabetic order): Szabolcs Boros, Marcus Brunner, David Durham, Dave Harrington, Jean-Philippe Martin-Flatin, George Pavlou, Robert Parhonyi, David Perkins, David Sidor, Andrea Westerinen and Bert Wijnen.

7. Normative References

[1] McCloghrie, K., Perkins, D. and J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[2] McCloghrie, K., Fine, M., Seligson, J., Chan, K., Hahn, S., Sahita, R., Smith, A. and F. Reichmeyer, "Structure of Policy Provisioning Information (SPPI)", RFC 3159, August 2001.

[3] Distributed Management Task Force, "Common Information Model (CIM) Specification Version 2.2", DSP 0004, June 1999.

[4] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[5] Object Management Group, "Unified Modeling Language (UML), Version 1.4", formal/2001-09-67, September 2001.

[6] International Organization for Standardization, "Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)", International Standard 8824, December 1987.

[7] International Telecommunication Union, "Information technology - Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Objects", Recommendation X.722, 1992.

8. Informative References

[8] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J. and S. Waldbusser, "Terminology for Policy-Based Management", RFC 3198, November 2001.

[9] Bernet, Y., Blake, S., Grossman, D. and A. Smith, "An Informal Management Model for Diffserv Routers", RFC 3290, May 2002.

[10] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

9. Authors' Addresses

Aiko Pras
University of Twente
PO Box 217
7500 AE Enschede
The Netherlands

Phone: +31 53 4893778
EMail: pras@ctit.utwente.nl

Juergen Schoenwaelder
University of Osnabrueck
Albrechtstr. 28
49069 Osnabrueck
Germany

Phone: +49 541 969-2483
EMail: schoenw@informatik.uni-osnabrueck.de

10. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

[x42.com] [rfc-index] [text only]

APPENDIX 3

RFC 3060

Updated by 3460

Network Working Group
Request for Comments: 3060
Category: Standards Track

B. Moore
IBM
E. Ellesson
LongBoard, Inc.
J. Strassner
A. Westerinen
Cisco Systems
February 2001

Policy Core Information Model -- Version 1 Specification

Status of this Memo

Copyright Notice

Abstract

This document presents the object-oriented information model for representing policy information developed jointly in the IETF Policy Framework WG and as extensions to the Common Information Model (CIM) activity in the Distributed Management Task Force (DMTF). This model defines two hierarchies of object classes: structural classes representing policy information and control of policies, and association classes that indicate how instances of the structural classes are related to each other. Subsequent documents will define mappings of this information model to various concrete implementations, for example, to a directory that uses LDAPv3 as its access protocol.

Table of Contents

```
   1. Introduction................................................. 4
   2. Modeling Policies............................................ 5
      2.1. Policy Scope............................................ 8
      2.2. Declarative versus Procedural Model..................... 8
   3. Overview of the Policy Core Information Model................ 10
   4. Inheritance Hierarchies for the Policy Core Information Model.. 13
      4.1. Implications of CIM Inheritance......................... 15
   5. Details of the Model......................................... 15
```

```
     5.1. Reusable versus Rule-Specific Conditions and Actions...... 15
     5.2. Roles..................................................... 17
     5.2.1. Roles and Role Combinations............................. 17
     5.2.2. The PolicyRoles Property................................ 21
     5.3. Local Time and UTC Time in PolicyTimePeriodConditions..... 21
     5.4. CIM Data Types............................................ 23
     5.5. Comparison between CIM and LDAP Class Specifications...... 24
   6. Class Definitions............................................. 25
     6.1. The Abstract Class "Policy"............................... 25
     6.1.1. The Property "CommonName (CN)".......................... 26
     6.1.2. The Multi-valued Property "PolicyKeywords".............. 26
     6.1.3. The Property "Caption" (Inherited from ManagedElement).. 27
     6.1.4. The Property "Description" (Inherited from
            ManagedElement)......................................... 27
     6.2. The Class "PolicyGroup"................................... 27
     6.3. The Class "PolicyRule".................................... 29
     6.3.1. The Property "Enabled".................................. 31
     6.3.2. The Property "ConditionListType"........................ 31
     6.3.3. The Property "RuleUsage"................................ 31
     6.3.4. The Property "Priority"................................. 32
     6.3.5. The Property "Mandatory"................................ 32
     6.3.6. The Property "SequencedActions"......................... 33
     6.3.7. The Multi-valued Property "PolicyRoles"................. 33
     6.4. The Abstract Class "PolicyCondition"...................... 34
     6.5. The Class "PolicyTimePeriodCondition"..................... 36
     6.5.1. The Property "TimePeriod"............................... 38
     6.5.2. The Property "MonthOfYearMask".......................... 39
     6.5.3. The Property "DayOfMonthMask"........................... 39
     6.5.4. The Property "DayOfWeekMask"............................ 40
     6.5.5. The Property "TimeOfDayMask"............................ 41
     6.5.6. The Property "LocalOrUtcTime"........................... 42
     6.6. The Class "VendorPolicyCondition"......................... 42
     6.6.1. The Multi-valued Property "Constraint".................. 43
     6.6.2. The Property "ConstraintEncoding"....................... 43
     6.7. The Abstract Class "PolicyAction"......................... 44
     6.8. The Class "VendorPolicyAction"............................ 45
     6.8.1. The Multi-valued Property "ActionData".................. 45
     6.8.2. The Property "ActionEncoding"........................... 46
     6.9. The Class "PolicyRepository".............................. 46
   7. Association and Aggregation Definitions....................... 46
     7.1. Associations.............................................. 47
     7.2. Aggregations.............................................. 47
     7.3. The Abstract Aggregation "PolicyComponent................. 47
     7.4. The Aggregation "PolicyGroupInPolicyGroup"................ 47
     7.4.1. The Reference "GroupComponent".......................... 48
     7.4.2. The Reference "PartComponent"........................... 48
     7.5. The Aggregation "PolicyRuleInPolicyGroup"................. 48
     7.5.1. The Reference "GroupComponent".......................... 49
```

```
      7.5.2. The Reference "PartComponent"........................... 49
      7.6. The Aggregation "PolicyConditionInPolicyRule"............. 49
      7.6.1. The Reference "GroupComponent"......................... 50
      7.6.2. The Reference "PartComponent".......................... 50
      7.6.3. The Property "GroupNumber"............................. 50
      7.6.4. The Property "ConditionNegated"........................ 51
      7.7. The Aggregation "PolicyRuleValidityPeriod"............... 51
      7.7.1. The Reference "GroupComponent"......................... 52
      7.7.2. The Reference "PartComponent".......................... 52
      7.8. The Aggregation "PolicyActionInPolicyRule"............... 52
      7.8.1. The Reference "GroupComponent"......................... 53
      7.8.2. The Reference "PartComponent".......................... 53
      7.8.3. The Property "ActionOrder"............................. 53
      7.9. The Abstract Association "PolicyInSystem"................ 54
      7.10. The Weak Association "PolicyGroupInSystem".............. 55
      7.10.1. The Reference "Antecedent"............................ 55
      7.10.2. The Reference "Dependent"............................. 55
      7.11. The Weak Association "PolicyRuleInSystem"............... 56
      7.11.1. The Reference "Antecedent"............................ 56
      7.11.2. The Reference "Dependent"............................. 56
      7.12. The Association "PolicyConditionInPolicyRepository"...... 56
      7.12.1. The Reference "Antecedent"............................ 57
      7.12.2. The Reference "Dependent"............................. 57
      7.13. The Association "PolicyActionInPolicyRepository"......... 57
      7.13.1. The Reference "Antecedent"............................ 58
      7.13.2. The Reference "Dependent"............................. 58
      7.14. The Aggregation "PolicyRepositoryInPolicyRepository"..... 58
      7.14.1. The Reference "GroupComponent"........................ 58
      7.14.2. The Reference "PartComponent"......................... 59
   8. Intellectual Property........................................ 59
   9. Acknowledgements............................................. 59
   10. Security Considerations..................................... 60
   11. References.................................................. 62
   12. Authors' Addresses.......................................... 64
   13. Appendix A:  Class Identification in a Native CIM
       Implementation.............................................. 65
     13.1. Naming Instances of PolicyGroup and PolicyRule........... 65
     13.1.1. PolicyGroup's CIM Keys................................. 65
     13.1.2. PolicyRule's CIM Keys.................................. 66
     13.2. Naming Instances of PolicyCondition and Its Subclasses... 67
     13.2.1. PolicyCondition's CIM Keys............................. 69
     13.3. Naming Instances of PolicyAction and Its Subclasses...... 71
     13.4. Naming Instances of PolicyRepository..................... 72
     13.5. Role of the CreationClassName Property in Naming......... 73
     13.6. Object References........................................ 73
   14. Appendix B:  The Core Policy MOF............................ 75
   15. Full Copyright Statement................................... 100
```

1. Introduction

This document presents the object-oriented information model for representing policy information currently under joint development in the IETF Policy Framework WG and as extensions to the Common Information Model (CIM) activity in the Distributed Management Task Force (DMTF). This model defines two hierarchies of object classes: structural classes representing policy information and control of policies, and association classes that indicate how instances of the structural classes are related to each other. Subsequent documents will define mappings of this information model to various concrete implementations, for example, to a directory that uses LDAPv3 as its access protocol. The components of the CIM schema are available via the following URL: http://www.dmtf.org/spec/cims.html [1].

The policy classes and associations defined in this model are sufficiently generic to allow them to represent policies related to anything. However, it is expected that their initial application in the IETF will be for representing policies related to QoS (DiffServ and IntServ) and to IPSec. Policy models for application-specific areas such as these may extend the Core Model in several ways. The preferred way is to use the PolicyGroup, PolicyRule, and PolicyTimePeriodCondition classes directly, as a foundation for representing and communicating policy information. Then, specific subclasses derived from PolicyCondition and PolicyAction can capture application-specific definitions of conditions and actions of policies.

Two subclasses, VendorPolicyCondition and VendorPolicyAction, are also included in this document, to provide a standard extension mechanism for vendor-specific extensions to the Policy Core Information Model.

This document fits into the overall framework for representing, deploying, and managing policies being developed by the Policy Framework Working Group. It traces its origins to work that was originally done for the Directory-enabled Networks (DEN) specification, reference [5]. Work on the DEN specification by the DEN Ad-Hoc Working Group itself has been completed. Further work to standardize the models contained in it will be the responsibility of selected working groups of the CIM effort in the Distributed Management Task Force (DMTF). DMTF standardization of the core policy model is the responsibility of the SLA Policy working group in the DMTF.

This document is organized in the following manner:

o Section 2 provides a general overview of policies and how they are modeled.

o Section 3 presents a high-level overview of the classes and associations comprising the Policy Core Information Model.

o The remainder of the document presents the detailed specifications for each of the classes and associations.

o Appendix A overviews naming for native CIM implementations. Other mappings, such as LDAPv3, will have their own naming mechanisms.

o Appendix B reproduces the DMTF's Core Policy MOF specification.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, reference [3].

2. Modeling Policies

The classes comprising the Policy Core Information Model are intended to serve as an extensible class hierarchy (through specialization) for defining policy objects that enable application developers, network administrators, and policy administrators to represent policies of different types.

One way to think of a policy-controlled network is to first model the network as a state machine and then use policy to control which state a policy-controlled device should be in or is allowed to be in at any given time. Given this approach, policy is applied using a set of policy rules. Each policy rule consists of a set of conditions and a set of actions. Policy rules may be aggregated into policy groups. These groups may be nested, to represent a hierarchy of policies.

The set of conditions associated with a policy rule specifies when the policy rule is applicable. The set of conditions can be expressed as either an ORed set of ANDed sets of condition statements or an ANDed set of ORed sets of statements. Individual condition statements can also be negated. These combinations are termed, respectively, Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF) for the conditions.

If the set of conditions associated with a policy rule evaluates to TRUE, then a set of actions that either maintain the current state of the object or transition the object to a new state may be executed.

For the set of actions associated with a policy rule, it is possible to specify an order of execution, as well as an indication of whether the order is required or merely recommended. It is also possible to indicate that the order in which the actions are executed does not matter.

Policy rules themselves can be prioritized. One common reason for doing this is to express an overall policy that has a general case with a few specific exceptions.

For example, a general QoS policy rule might specify that traffic originating from members of the engineering group is to get Bronze Service. A second policy rule might express an exception: traffic originating from John, a specific member of the engineering group, is to get Gold Service. Since traffic originating from John satisfies the conditions of both policy rules, and since the actions associated with the two rules are incompatible, a priority needs to be established. By giving the second rule (the exception) a higher priority than the first rule (the general case), a policy administrator can get the desired effect: traffic originating from John gets Gold Service, and traffic originating from all the other members of the engineering group gets Bronze Service.

Policies can either be used in a stand-alone fashion or aggregated into policy groups to perform more elaborate functions. Stand-alone policies are called policy rules. Policy groups are aggregations of policy rules, or aggregations of policy groups, but not both. Policy groups can model intricate interactions between objects that have complex interdependencies. Examples of this include a sophisticated user logon policy that sets up application access, security, and reconfigures network connections based on a combination of user identity, network location, logon method and time of day. A policy group represents a unit of reusability and manageability in that its management is handled by an identifiable group of administrators and its policy rules would be consistently applied

Stand-alone policies are those that can be expressed in a simple statement. They can be represented effectively in schemata or MIBs. Examples of this are VLAN assignments, simple YES/NO QoS requests, and IP address allocations. A specific design goal of this model is to support both stand-alone and aggregated policies.

Policy groups and rules can be classified by their purpose and intent. This classification is useful in querying or grouping policy rules. It indicates whether the policy is used to motivate when or how an action occurs, or to characterize services (that can then be used, for example, to bind clients to network services). Describing each of these concepts in more detail,

- o Motivational Policies are solely targeted at whether or how a policy's goal is accomplished. Configuration and Usage Policies are specific kinds of Motivational Policies. Another example is the scheduling of file backup based on disk write activity from 8am to 3pm, M-F.

- o Configuration Policies define the default (or generic) setup of a managed entity (for example, a network service). Examples of Configuration Policies are the setup of a network forwarding service or a network-hosted print queue.

- o Installation Policies define what can and cannot be put on a system or component, as well as the configuration of the mechanisms that perform the install. Installation policies typically represent specific administrative permissions, and can also represent dependencies between different components (e.g., to complete the installation of component A, components B and C must be previously successfully installed or uninstalled).

- o Error and Event Policies. For example, if a device fails between 8am and 9pm, call the system administrator, otherwise call the Help Desk.

- o Usage Policies control the selection and configuration of entities based on specific "usage" data. Configuration Policies can be modified or simply re-applied by Usage Policies. Examples of Usage Policies include upgrading network forwarding services after a user is verified to be a member of a "gold" service group, or reconfiguring a printer to be able to handle the next job in its queue.

- o Security Policies deal with verifying that the client is actually who the client purports to be, permitting or denying access to resources, selecting and applying appropriate authentication mechanisms, and performing accounting and auditing of resources.

- o Service Policies characterize network and other services (not use them). For example, all wide-area backbone interfaces shall use a specific type of queuing.

  Service policies describe services available in the network. Usage policies describe the particular binding of a client of the network to services available in the network.

These categories are represented in the Policy Core Information Model by special values defined for the PolicyKeywords property of the abstract class Policy.

2.1. Policy Scope

Policies represent business goals and objectives. A translation must be made between these goals and objectives and their realization in the network. An example of this could be a Service Level Agreement (SLA), and its objectives and metrics (Service Level Objectives, or SLOs), that are used to specify services that the network will provide for a given client. The SLA will usually be written in high-level business terminology. SLOs address more specific metrics in support of the SLA. These high-level descriptions of network services and metrics must be translated into lower-level, but also vendor-and device-independent specifications. The Policy Core Information Model classes are intended to serve as the foundation for these lower-level, vendor- and device-independent specifications.

It is envisioned that the definition of the Policy Core Informational Model in this document is generic in nature and is applicable to Quality of Service (QoS), to non-QoS networking applications (e.g., DHCP and IPSec), and to non-networking applications (e.g., backup policies, auditing access, etc.).

2.2. Declarative versus Procedural Model

The design of the Policy Core Information Model is influenced by a declarative, not procedural, approach. More formally, a declarative language is used to describe relational and functional languages. Declarative languages describe relationships between variables in terms of functions or inference rules, to which the interpreter or compiler can apply a fixed algorithm in order to produce a result. An imperative (or procedural) language specifies an explicit sequence of steps to follow in order to produce a result.

It is important to note that this information model does not rule out the use of procedural languages. Rather, it recognizes that both declarative as well as procedural languages can be used to implement policy. This information model is better viewed as being declarative because the sequence of steps for doing the processing of declarative statements tends to be left to the implementer. However, we have provided the option of expressing the desired order of action execution in this policy information model, and for expressing whether the order is mandatory or not. In addition, rather than trying to define algorithms or sets of instructions or steps that must be followed by a policy rule, we instead define a set of modular building blocks and relationships that can be used in a declarative or procedural fashion to define policies.

Compare this to a strictly procedural model. Taking such an approach would require that we specify the condition testing sequence, and the action execution sequence, in the policy repository itself. This would, indeed, constrain the implementer. This is why the policy model is characterized as a declarative one. That is, the information model defines a set of attributes, and a set of entities that contain these attributes. However, it does NOT define either the algorithm to produce a result using the attributes or an explicit sequence of steps to produce a result.

There are several design considerations and trade-offs to make in this respect.

1. On the one hand, we would like a policy definition language to be reasonably human-friendly for ease of definitions and diagnostics. On the other hand, given the diversity of devices (in terms of their processing capabilities) which could act as policy decision points, we would like to keep the language somewhat machine-friendly. That is, it should be relatively simple to automate the parsing and processing of the language in network elements. The approach taken is to provide a set of classes and attributes that can be combined in either a declarative or procedural approach to express policies that manage network elements and services. The key point is to avoid trying to standardize rules or sets of steps to be followed in defining a policy. These must be left up to an implementation. Interoperability is achieved by standardizing the building blocks that are used to represent policy data and information.

2. An important decision to make is the semantic style of the representation of the information.

   The declarative approach that we are describing falls short of being a "true" declarative model. Such a model would also specify the algorithms used to combine the information and policy rules to achieve particular behavior. We avoid specifying algorithms for the same reason that we avoid specifying sets of steps to be followed in a policy rule. However, the design of the information model more closely follows that of a declarative language, and may be easier to understand if such a conceptual model is used. This leads to our third point, acknowledging a lack of "completeness" and instead relying on presenting information that the policy processing entity will work with.

3. It is important to control the complexity of the specification, trading off richness of expression of data in the core information model for ease of implementation and use. It is important to acknowledge the collective lack of experience in the field

regarding policies to control and manage network services and hence avoid the temptation of aiming for "completeness". We should instead strive to facilitate definition of a set of common policies that customers require today (e.g., VPN and QoS) and allow migration paths towards supporting complex policies as customer needs and our understanding of these policies evolve with experience. Specifically, in the context of the declarative style language discussed above, it is important to avoid having full blown predicate calculus as the language, as it would render many important problems such as consistency checking and policy decision point algorithms intractable. It is useful to consider a reasonably constrained language from these perspectives.

The Policy Core Information Model strikes a balance between complexity and lack of power by using the well understood logical concepts of Disjunctive Normal Form and Conjunctive Normal Form for combining simple policy conditions into more complex ones.

3. Overview of the Policy Core Information Model

The following diagram provides an overview of the five central classes comprising the Policy Core Information Model, their associations to each other, and their associations to other classes in the overall CIM model. Note that the abstract class Policy and the two extension classes VendorPolicyCondition and VendorPolicyAction are not shown.

NOTE: For cardinalities, "*" is an abbreviation for "0..n".

```
                          +-----------+
                          |  System   |
       .....              +--^-----^--+       .....
       .   .                1.    1.          .   .
      *.(a).*                .(b)  .(c)      *.(d).*
    +--v---v---------+       .     .        +-v---v------------+
    |  PolicyGroup   <........     .        | PolicyRepository |
    |                | w *         .        |                  |
    +------^---------+             .        +-----^---------^--+
          *.                       .         0..1 .    0..1 .
           .(e)                    .              .(f)      .(g)
          *.                       .              .         .
    +------v------+ w *            .              .         .
    |             <.................              .         .
    | PolicyRule  |                               .         .
    |             |                               .         .
    |             |                               .         .
    |             <........................       .         .
    |             |*      (h)             .       .         .
    |             |                       .       .         .
    |             |                       .       .         .
    |             |                       .       .         .
    |             |                       .       .         .
    |             |                       .       .         .
    |             |                       .       .         .
    |             |                       .*      .*        .
    |             |            +----------v-------v--+      .
    |             |            |  PolicyCondition    |      .
    |             |           *+---------------------+      .
    |             |       (i)             ^                 .
    |             <..............         I                 .
    |             |*            .         I                 .
    |             |             .*        ^                 .
    |             |        +----v----------------------+    .
    |             |        | PolicyTimePeriodCondition |    .
    |             |        +---------------------------+    .
    |             |       (j)                               .
    |             <........................                 .
    |             |*                      .                 .
    |             |                       .*                .
    |             |            +----------v---------+*      .
    |             |            | PolicyAction       <........
    +-------------+            +--------------------+
```

Figure 1.   Overview of the Core Policy Classes and Relationships

In this figure the boxes represent the classes, and the dotted arrows represent the associations. The following associations appear:

(a) PolicyGroupInPolicyGroup

(b) PolicyGroupInSystem

(c) PolicyRuleInSystem

(d) PolicyRepositoryInPolicyRepository

(e) PolicyRuleInPolicyGroup

(f) PolicyConditionInPolicyRepository

(g) PolicyActionInPolicyRepository

(h) PolicyConditionInPolicyRule

(i) PolicyRuleValidityPeriod

(j) PolicyActionInPolicyRule

An association always connects two classes. The "two" classes may, however, be the same class, as is the case with the PolicyGroupInPolicyGroup association, which represents the recursive containment of PolicyGroups in other PolicyGroups. The PolicyRepositoryInPolicyRepository association is recursive in the same way.

An association includes cardinalities for each of the related classes. These cardinalities indicate how many instances of each class may be related to an instance of the other class. For example, the PolicyRuleInPolicyGroup association has the cardinality range "*' (that is, "0..n") for both the PolicyGroup and PolicyRule classes. These ranges are interpreted as follows:

o The "*" written next to PolicyGroup indicates that a PolicyRule may be related to no PolicyGroups, to one PolicyGroup, or to more than one PolicyGroup via the PolicyRuleInPolicyGroup association. In other words, a PolicyRule may be contained in no PolicyGroups, in one PolicyGroups, or in more than one PolicyGroup.

o The "*" written next to PolicyRule indicates that a PolicyGroup may be related to no PolicyRules, to one PolicyRule, or to more than one PolicyRule via the PolicyRuleInPolicyGroup association. In other words, a PolicyGroup may contain no PolicyRules, one PolicyRule, or more than one PolicyRule.

The "w" written next to the PolicyGroupInSystem and PolicyRuleInSystem indicates that these are what CIM terms "aggregations with weak references", or more briefly, "weak aggregations". A weak aggregation is simply an indication of a naming scope. Thus these two aggregations indicate that an instance of a PolicyGroup or PolicyRule is named within the scope of a System object. A weak aggregation implicitly has the cardinality 1..1 at the end opposite the 'w'.

The associations shown in Figure 1 are discussed in more detail in Section 7.

4. Inheritance Hierarchies for the Policy Core Information Model

The following diagram illustrates the inheritance hierarchy for the core policy classes:

```
ManagedElement (abstract)
 |
 +--Policy (abstract)
 |  |
 |  +---PolicyGroup
 |  |
 |  +---PolicyRule
 |  |
 |  +---PolicyCondition (abstract)
 |  |          |
 |  |          +---PolicyTimePeriodCondition
 |  |          |
 |  |          +---VendorPolicyCondition
 |  |
 |  +---PolicyAction (abstract)
 |             |
 |             +---VendorPolicyAction
 |
 +--ManagedSystemElement (abstract)
    |
    +--LogicalElement (abstract)
       |
       +--System (abstract)
          |
          +--AdminDomain (abstract)
             |
             +---PolicyRepository
```

Figure 2. Inheritance Hierarchy for the Core Policy Classes

ManagedElement, ManagedSystemElement, LogicalElement, System, and AdminDomain are defined in the CIM schema [1]. These classes are not discussed in detail in this document.

In CIM, associations are also modeled as classes. For the Policy Core Information Model, the inheritance hierarchy for the associations is as follows:

```
[unrooted]
 |
 +---PolicyComponent (abstract)
 |    |
 |    +---PolicyGroupInPolicyGroup
 |    |
 |    +---PolicyRuleInPolicyGroup
 |    |
 |    +---PolicyConditionInPolicyRule
 |    |
 |    +---PolicyRuleValidityPeriod
 |    |
 |    +---PolicyActionInPolicyRule
 |
 +---Dependency (abstract)
 |    |
 |    +---PolicyInSystem (abstract)
 |         |
 |         +---PolicyGroupInSystem
 |         |
 |         +---PolicyRuleInSystem
 |         |
 |         +---PolicyConditionInPolicyRepository
 |         |
 |         +---PolicyActionInPolicyRepository
 |
 +---Component (abstract)
      |
      +---SystemComponent
           |
           +---PolicyRepositoryInPolicyRepository
```

Figure 3. Inheritance Hierarchy for the Core Policy Associations

The Dependency, Component, and SystemComponent associations are defined in the CIM schema [1], and are not discussed further in this document.

4.1. Implications of CIM Inheritance

From the CIM schema, both properties and associations are inherited to the Policy classes. For example, the class ManagedElement is referenced in the associations Dependency, Statistics and MemberOfCollection. And, the Dependency association is in turn referenced in the DependencyContext association. At this very abstract and high level in the inheritance hierarchy, the number of these associations is very small and their semantics are quite general.

Many of these inherited associations convey additional semantics that are not needed in understanding the Policy Core Information Model. In fact, they are defined as OPTIONAL in the CIM Schema - since their cardinality is "0..n" on all references. The PCIM document specifically discusses what is necessary to support and instantiate. For example, through subclassing of the Dependency association, the exact Dependency semantics in PCIM are described.

So, one may wonder what to do with these other inherited associations. The answer is "ignore them unless you need them". You would need them to describe additional information and semantics for policy data. For example, it may be necessary to capture statistical data for a PolicyRule (either for the rule in a repository or for when it is executing in a policy system). Some examples of statistical data for a rule are the number of times it was downloaded, the number of times its conditions were evaluated, and the number of times its actions were executed. (These types of data would be described in a subclass of CIM_StatisticalInformation.) In these cases, the Statistics association inherited from ManagedElement to PolicyRule may be used to describe the tie between an instance of a PolicyRule and the set of statistics for it.

5. Details of the Model

The following subsections discuss several specific issues related to the Policy Core Information Model.

5.1. Reusable versus Rule-Specific Conditions and Actions

Policy conditions and policy actions can be partitioned into two groups: ones associated with a single policy rule, and ones that are reusable, in the sense that they may be associated with more than one policy rule. Conditions and actions in the first group are termed "rule-specific" conditions and actions; those in the second group are characterized as "reusable".

It is important to understand that the difference between a rule-specific condition or action and a reusable one is based on the intent of the policy administrator for the condition or action, rather than on the current associations in which the condition or action participates. Thus a reusable condition or action (that is, one that a policy administrator has created to be reusable) may at some point in time be associated with exactly one policy rule, without thereby becoming rule-specific.

There is no inherent difference between a rule-specific condition or action and a reusable one. There are, however, differences in how they are treated in a policy repository. For example, it's natural to make the access permissions for a rule-specific condition or action identical to those for the rule itself. It's also natural for a rule-specific condition or action to be removed from the policy repository at the same time the rule is. With reusable conditions and actions, on the other hand, access permissions and existence criteria must be expressible without reference to a policy rule.

The preceding paragraph does not contain an exhaustive list of the ways in which reusable and rule-specific conditions should be treated differently. Its purpose is merely to justify making a semantic distinction between rule-specific and reusable, and then reflecting this distinction in the policy model itself.

An issue is highlighted by reusable and rule-specific policy conditions and policy actions: the lack of a programmatic capability for expressing complex constraints involving multiple associations. Taking PolicyCondition as an example, there are two aggregations to look at. PolicyConditionInPolicyRule has the cardinality * at both ends, and PolicyConditionInPolicyRepository has the cardinality * at the PolicyCondition end, and [0..1] at the PolicyRepository end.

Globally, these cardinalities are correct. However, there's more to the story, which only becomes clear if we examine the cardinalities separately for the two cases of a rule-specific PolicyCondition and a reusable one.

For a rule-specific PolicyCondition, the cardinality of PolicyConditionInPolicyRule at the PolicyRule end is [1..1], rather than [0..n] (recall that * is an abbreviation for [0..n]), since the condition is unique to one policy rule. And the cardinality of PolicyConditionInPolicyRepository at the PolicyRepository end is [0..0], since the condition is not in the "re-usable" repository. This is OK, since these are both subsets of the specified cardinalities.

For a reusable PolicyCondition, however, the cardinality of PolicyConditionInPolicyRepository at the PolicyRepository end is [1..1], since the condition must be in the repository. And, the cardinality of PolicyConditionInPolicyRule at the PolicyRule end is [0..n]. This last point is important: a reusable PolicyCondition may be associated with 0, 1, or more than 1 PolicyRules, via exactly the same association PolicyConditionInPolicyRule that binds a rule-specific condition to its PolicyRule.

Currently the only way to document constraints of this type is textually. More formal methods for documenting complex constraints are needed.

5.2. Roles

5.2.1. Roles and Role Combinations

The concept of role is central to the design of the entire Policy Framework. The idea behind roles is a simple one. Rather than configuring, and then later having to update the configuration of, hundreds or thousands (or more) of resources in a network, a policy administrator assigns each resource to one or more roles, and then specifies the policies for each of these roles. The Policy Framework is then responsible for configuring each of the resources associated with a role in such a way that it behaves according to the policies specified for that role. When network behavior must be changed, the policy administrator can perform a single update to the policy for a role, and the Policy Framework will ensure that the necessary configuration updates are performed on all the resources playing that role.

A more formal definition of a role is as follows:

> A role is a type of attribute that is used to select one or more policies for a set of entities and/or components from among a much larger set of available policies.

Roles can be combined together. Here is a formal definition of a "role- combination":

> A role-combination is a set of attributes that are used to select one or more policies for a set of entities and/or components from among a much larger set of available policies. As the examples below illustrate, the selection process for a role combination chooses policies associated with the combination itself, policies associated with each of its sub-combinations, and policies associated with each of the individual roles in the role-combination.

It is important to note that a role is more than an attribute. A role defines a particular function of an entity or component that can be used to identify particular behavior associated with that entity or component. This difference is critical, and is most easily understood by thinking of a role as a selector. When used in this manner, one role (or role-combination) selects a different set of policies than a different role (or role-combination) does.

Roles and role-combinations are especially useful in selecting which policies are applicable to a particular set of entities or components when the policy repository can store thousands or hundreds of thousands of policies. This use emphasizes the ability of the role (or role- combination) to select the small subset of policies that are applicable from a huge set of policies that are available.

An example will illustrate how role-combinations actually work. Suppose an installation has three roles defined for interfaces: "Ethernet", "Campus", and "WAN". In the Policy Repository, some policy rules could be associated with the role "Ethernet"; these rules would apply to all Ethernet interfaces, regardless of whether they were on the campus side or the WAN side. Other rules could be associated with the role-combination "Campus"+"Ethernet"; these rules would apply to the campus-side Ethernet interfaces, but not to those on the WAN side. Finally, a third set of rules could be associated with the role-combination "Ethernet"+"WAN"; these rules would apply to the WAN-side Ethernet interfaces, but not to those on the campus side. (The roles in a role-combination appear in alphabetical order in these examples, because that is how they appear in the information model.)

If we have a specific interface A that's associated with the role-combination "Ethernet"+"WAN", we see that it should have three categories of policy rules applied to it: those for the "Ethernet" role, those for the "WAN" role, and those for the role-combination "Ethernet"+"WAN". Going one step further, if interface B is associated with the role- combination "branch-office"+"Ethernet"+"WAN", then B should have seven categories of policy rules applied to it - those associated with the following role-combinations:

- o "branch-office"
- o "Ethernet"
- o "WAN"
- o "branch-office"+"Ethernet"
- o "branch-office"+"WAN"
- o "Ethernet"+"WAN"
- o "branch-office"+"Ethernet"+"WAN".

In order to get all of the right policy rules for a resource like interface B, a PDP must expand the single role-combination it receives for B into this list of seven role-combinations, and then retrieve from the Policy Repository the corresponding seven sets of policy rules. Of course this example is unusually complicated: the normal case will involve expanding a two-role combination into three values identifying three sets of policy rules.

Role-combinations also help to simplify somewhat the problem of identifying conflicts between policy rules. With role-combinations, it is possible for a policy administrator to specify one set of policy rules for campus-side Ethernet interfaces, and a second set of policy rules for WAN-side Ethernet interfaces, without having to worry about conflicts between the two sets of rules. The policy administrator simply "turns off" conflict detection for these two sets of rules, by telling the policy management system that the roles "Campus" and "WAN" are incompatible with each other. This indicates that the role combination will never occur, and therefore conflicts will never occur. In some cases the technology itself might identify incompatible roles: "Ethernet" and "FrameRelay", for example. But for less precise terms like "Campus" and "WAN", the policy administrator must say whether they identify incompatible roles.

When the policy administrator does this, there are three effects:

1. If an interface has assigned to it a role-combination involving both "Campus" and "WAN", then the policy management system can flag it as an error.

2. If a policy rule is associated with a role-combination involving both "Campus" and "WAN", then the policy management system can flag it as an error.

3. If the policy management system sees two policy rules, where one is tied to the role "Campus" (or to a role-combination that includes the role "Campus") and the other is tied to the role "WAN" (or to a role- combination that includes the role "WAN"), then the system does not need to look for conflicts between the two policy rules: because of the incompatible roles, the two rules cannot possibly conflict.

```
                    +-------------------+
                    | Policy Repository |
                    +-------------------+
                             V
                             V retrieval of policy
                             V
                        +---------+
                        | PDP/PEP |
                        +---------+
                             v
                             v application of policy
                             v
                      +----------------+
                      | Network Entity |
                      +----------------+
```

Figure 4. Retrieval and Application of a Policy

Figure 4, which is introduced only as an example of how the Policy Framework might be implemented by a collection of network components, illustrates how roles operate within the Policy Framework. Because the distinction between them is not important to this discussion, the PDP and the PEP are combined in one box. The points illustrated here apply equally well, though, to an environment where the PDP and the PEP are implemented separately.

A role represents a functional characteristic or capability of a resource to which policies are applied. Examples of roles include Backbone interface, Frame Relay interface, BGP-capable router, web server, firewall, etc. The multiple roles assigned to a single resource are combined to form that resource's role combination. Role combinations are represented in the PCIM by values of the PolicyRoles property in the PolicyRule class. A PDP uses policy roles as follows to identify the policies it needs to be aware of:

1. The PDP learns in some way the list of roles that its PEPs play. This information might be configured at the PDP, the PEPs might supply it to the PDP, or the PDP might retrieve it from a repository.

2. Using repository-specific means, the PDP determines where to look for policy rules that might apply to it.

3. Using the roles and role-combinations it received from its PEPs as indicated in the examples above, the PDP is able to locate and retrieve the policy rules that are relevant to it.

5.2.2. The PolicyRoles Property

As indicated earlier, PolicyRoles is a property associated with a policy rule. It is an array holding "role combinations" for the policy rule, and correlates with the roles defined for a network resource. Using the PolicyRoles property, it is possible to mark a policy rule as applying, for example, to a Frame Relay interface or to a backbone ATM interface. The PolicyRoles property take strings of the form:

```
<RoleName>[&&<RoleName>]*
```

Each value of this property represents a role combination, including the special case of a "combination" containing only one role. As the format indicates, the role names in a role combination are ANDed together to form a single selector. The multiple values of the PolicyRoles property are logically ORed, to make it possible for a policy rule to have multiple selectors.

The individual role names in a role combination must appear in alphabetical order (according to the collating sequence for UCS-2 characters), to make the string matches work correctly. The role names used in an environment are specified by the policy administrator.

5.3. Local Time and UTC Time in PolicyTimePeriodConditions

An instance of PolicyTimePeriodCondition has up to five properties that represent times: TimePeriod, MonthOfYearMask, DayOfMonthMask, DayOfWeekMask, and TimeOfDayMask. All of the time-related properties in an instance of PolicyTimePeriodCondition represent one of two types of times: local time at the place where a policy rule is applied, or UTC time. The property LocalOrUtcTime indicates which time representation applies to an instance of PolicyTimePeriodCondition.

Since the PCIM provides only for local time and UTC time, a Policy Management Tool that provides for other time representations (for example, a fixed time at a particular location) will need to map from these other representations to either local time or UTC time. An example will illustrate the nature of this mapping.

Suppose a policy rule is tied to the hours of operation for a Help Desk: 0800 to 2000 Monday through Friday [US] Eastern Time. In order to express these times in PolicyTimePeriodCondition, a management tool must convert them to UTC times. (They are not local times, because they refer to a single time interval worldwide, not to intervals tied to the local clocks at the locations where the

PolicyRule is being applied.) As reference [10] points out, mapping from [US] Eastern Time to UTC time is not simply a matter of applying an offset: the offset between [US] Eastern Time and UTC time switches between -0500 and -0400 depending on whether Daylight Savings Time is in effect in the US.

Suppose the policy administrator's goal is to have a policy rule be valid from 0800 until 1200 [US] Eastern Time on every Monday, within the overall time period from the beginning of 2000 until the end of 2001. The Policy Management Tool could either be configured with the definition of what [US] Eastern Time means, or it could be configured with knowledge of where to go to get this information. Reference [10] contains further discussion of time zone definitions and where they might reside.

Armed with knowledge about [US] Eastern Time, the Policy Management Tool would create however many instances of PolicyTimePeriodCondition it needed to represent the desired intervals. Note that while there is an increased number of PolicyTimePeriodCondition instances, there is still just one PolicyRule, which is tied to all the PolicyTimePeriodCondition instances via the aggregation PolicyRuleValidityPeriod. Here are the first two of these instances:

```
1. TimePeriod:  20000101T050000/20000402T070000
   DayOfWeekMask:  { Monday }
   TimeOfDayMask:  T130000/T170000
   LocalOrUtcTime:  UTC

2. TimePeriod:  20000402T070000/20001029T070000
   DayOfWeekMask:  { Monday }
   TimeOfDayMask:  T120000/T160000
   LocalOrUtcTime:  UTC
```

There would be three more similar instances, for winter 2000-2001, summer 2001, and winter 2001 up through December 31.

Had the example been chosen differently, there could have been even more instances of PolicyTimePeriodCondition. If, for example, the

time interval had been from 0800 - 2200 [US] Eastern Time on Mondays, instance 1 above would have split into two instances: one with a UTC time interval of T130000/T240000 on Mondays, and another with a UTC time interval of T000000/T030000 on Tuesdays. So the end result would have been ten instances of PolicyTimePeriodCondition, not five.

By restricting PolicyTimePeriodCondition to local time and UTC time, the PCIM places the difficult and expensive task of mapping from "human" time representations to machine-friendly ones in the Policy

Management Tool. Another approach would have been to place in PolicyTimePeriodCondition a means of representing a named time zone, such as [US] Eastern Time. This, however, would have passed the difficult mapping responsibility down to the PDPs and PEPs. It is better to have a mapping such as the one described above done once in a Policy Management Tool, rather than having it done over and over in each of the PDPs (and possibly PEPs) that need to apply a PolicyRule.

5.4. CIM Data Types

Since PCIM extends the CIM Schema, a correspondence between data types used in both CIM and PCIM is needed. The following CIM data types are used in the class definitions that follow in Sections 6 and 7:

o uint8	unsigned 8-bit integer
o uint16	unsigned 16-bit integer
o boolean	Boolean
o string	UCS-2 string.

Strings in CIM are stored as UCS-2 characters, where each character is encoded in two octets. Thus string values may need to be converted when moving between a CIM environment and one that uses a different string encoding. For example, in an LDAP-accessible directory, attributes of type DirectoryString are stored in UTF-8 format. RFC 2279 [7] explains how to convert between these two formats.

When it is applied to a CIM string, a MaxLen value refers to the maximum number of characters in the string, rather than to the maximum number of octets.

In addition to the CIM data types listed above, the association classes in Section 7 use the following type:

o <classname> ref	strongly typed reference.

There is one obvious omission from this list of CIM data types: octet strings. This is because CIM treats octet strings as a derived data type. There are two forms of octet strings in CIM - an ordered uint8 array for single-valued strings, and a string array for multi-valued properties. Both are described by adding an "OctetString" qualifier (meta-data) to the property. This qualifier functions exactly like an SMIv2 (SNMP) Textual Convention, refining the syntax and semantics of the existing CIM data type.

The first four numeric elements of both of the "OctetString" representations are a length field. (The reason that the "numeric" adjective is added to the previous sentence is that the string property also includes '0' and 'x', as its first characters.) In both cases, these 4 numeric elements (octets) are included in calculating the length. For example, a single-valued octet string property having the value X'7C' would be represented by the uint8 array, X'00 00 00 05 7C'.

The strings representing the individual values of a multi-valued property qualified with the "OctetString" qualifier are constructed similarly:

1. Take a value to be encoded as an octet string (we'll use X'7C' as above), and prepend to it a four-octet length. The result is the same, X'00 00 00 05 7C'.

2. Convert this to a character string by introducing '0' and 'x' at the front, and removing all white space. Thus we have the 12-character string "0x000000057C". This string is the value of one of the array elements in the CIM string array. Since CIM uses the UCS-2 character set, it will require 24 octets to encode this 12-character string.

Mappings of the PCIM to particular data models are not required to follow this CIM technique of representing multi-valued octet strings as length- prefixed character strings. In an LDAP mapping, for example, it would be much more natural to simply use the Octet String syntax, and omit the prepended length octets.

5.5. Comparison between CIM and LDAP Class Specifications

There are a number of differences between CIM and LDAP class specifications. The ones that are relevant to the abbreviated class specifications in this document are listed below. These items are included here to help introduce the IETF community, which is already familiar with LDAP, to CIM modeling, and by extension, to information modeling in general.

o Instead of LDAP's three class types (abstract, auxiliary, structural), CIM has only two: abstract and instantiable. The type of a CIM class is indicated by the Boolean qualifier ABSTRACT.

o CIM uses the term "property" for what LDAP terms an "attribute".

- o CIM uses the array notation "[ ]" to indicate that a property is multi-valued. CIM defines three types of arrays: bags (contents are unordered, duplicates allowed), ordered bags (contents are ordered but duplicates are allowed) and indexed arrays (contents are ordered and no duplicates are allowed).

- o CIM classes and properties are identified by name, not by OID.

- o CIM classes use a different naming scheme for native implementations, than LDAP. The CIM naming scheme is documented in Appendix A since it is not critical to understanding the information model, and only applies when communicating with a native CIM implementation.

- o In LDAP, attribute definitions are global, and the same attribute may appear in multiple classes. In CIM, a property is defined within the scope of a single class definition. The property may be inherited into subclasses of the class in which it is defined, but otherwise it cannot appear in other classes. One side effect of this difference is that CIM property names tend to be much shorter than LDAP attribute names, since they are implicitly scoped by the name of the class in which they are defined.

There is also a notational convention that this document follows, to improve readability. In CIM, all class and property names are prefixed with the characters "CIM_". These prefixes have been omitted throughout this document, with one exception regarding naming, documented in Appendix A.

For the complete definition of the CIM specification language, see reference [2].

6. Class Definitions

The following sections contain the definitions of the PCIM classes.

6.1. The Abstract Class "Policy"

The abstract class Policy collects several properties that may be included in instances of any of the Core Policy classes (or their subclasses). For convenience, the two properties that Policy inherits from ManagedElement in the CIM schema are shown here as well.

The class definition is as follows:

```
   NAME             Policy
   DESCRIPTION      An abstract class with four properties for
                    describing a policy-related instance.
   DERIVED FROM     ManagedElement
   ABSTRACT         TRUE
   PROPERTIES       CommonName (CN)
                    PolicyKeywords[ ]
                           // Caption (inherited)
                           // Description (inherited)
```

6.1.1. The Property "CommonName (CN)"

The CN, or CommonName, property corresponds to the X.500 attribute commonName (cn). In X.500 this property specifies one or more user-friendly names (typically only one name) by which an object is commonly known, names that conform to the naming conventions of the country or culture with which the object is associated. In the CIM model, however, the CommonName property is single-valued.

```
   NAME             CN
   DESCRIPTION      A user-friendly name of a policy-related object.
   SYNTAX           string
```

6.1.2. The Multi-valued Property "PolicyKeywords"

This property provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- o Keywords defined in this document, or in documents that define subclasses of the classes defined in this document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- o Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

This document defines the following keywords: "UNKNOWN", "CONFIGURATION", "USAGE", "SECURITY", "SERVICE", "MOTIVATIONAL", "INSTALLATION", and "EVENT". These concepts were defined earlier in Section 2.

One additional keyword is defined: "POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subclasses of the Policy Core Information Model classes SHOULD define additional keywords to characterize instances of these subclasses. By convention, keywords defined in conjunction with class definitions are in uppercase. Installation-defined keywords can be in any case.

The property definition is as follows:

```
NAME           PolicyKeywords
DESCRIPTION    A set of keywords for characterizing /categorizing
               policy objects.
SYNTAX         string
```

6.1.3. The Property "Caption" (Inherited from ManagedElement)

This property provides a one-line description of a policy-related object.

```
NAME           Caption
DESCRIPTION    A one-line description of this policy-related object.
SYNTAX         string
```

6.1.4. The Property "Description" (Inherited from ManagedElement)

This property provides a longer description than that provided by the caption property.

```
NAME           Description
DESCRIPTION    A long description of this policy-related object.
SYNTAX         string
```

6.2. The Class "PolicyGroup"

This class is a generalized aggregation container. It enables either PolicyRules or PolicyGroups to be aggregated in a single container. Loops, including the degenerate case of a PolicyGroup that contains itself, are not allowed when PolicyGroups contain other PolicyGroups.

PolicyGroups and their nesting capabilities are shown in Figure 5 below. Note that a PolicyGroup can nest other PolicyGroups, and there is no restriction on the depth of the nesting in sibling PolicyGroups.

```
+------------------------------------------------------+
|                     PolicyGroup                      |
|                                                      |
| +---------------------+      +------------------+   |
| |    PolicyGroup A    |      |  PolicyGroup X   |   |
| |                     |      |                  |   |
| | +-----------------+ | ooo  |                  |   |
| | | PolicyGroup A1  | |      |                  |   |
| | +-----------------+ |      |                  |   |
| +---------------------+      +------------------+   |
+------------------------------------------------------+
```

Figure 5. Overview of the PolicyGroup class

As a simple example, think of the highest level PolicyGroup shown in Figure 5 above as a logon policy for US employees of a company. This PolicyGroup may be called USEmployeeLogonPolicy, and may aggregate several PolicyGroups that provide specialized rules per location. Hence, PolicyGroup A in Figure 5 above may define logon rules for employees on the West Coast, while another PolicyGroup might define logon rules for the Midwest (e.g., PolicyGroup X), and so forth.

Note also that the depth of each PolicyGroup does not need to be the same. Thus, the WestCoast PolicyGroup might have several additional layers of PolicyGroups defined for any of several reasons (different locales, number of subnets, etc..). The PolicyRules are therefore contained at n levels from the USEmployeeLogonPolicyGroup. Compare this to the Midwest PolicyGroup (PolicyGroup X), which might directly contain PolicyRules.

The class definition for PolicyGroup is as follows:

NAME	PolicyGroup
DESCRIPTION	A container for either a set of related PolicyRules or a set of related PolicyGroups.
DERIVED FROM	Policy
ABSTRACT	FALSE
PROPERTIES	NONE

No properties are defined for this class since it inherits all its properties from Policy. The class exists to aggregate PolicyRules or other PolicyGroups. It is directly instantiable. In an implementation, various key/identification properties MUST be defined. The keys for a native CIM implementation are defined in Appendix A, Section 13.1.1. Keys for an LDAP implementation will be defined in the LDAP mapping of this information model [11].

6.3. The Class "PolicyRule"

This class represents the "If Condition then Action" semantics associated with a policy. A PolicyRule condition, in the most general sense, is represented as either an ORed set of ANDed conditions (Disjunctive Normal Form, or DNF) or an ANDed set of ORed conditions (Conjunctive Normal Form, or CNF). Individual conditions may either be negated (NOT C) or unnegated (C). The actions specified by a PolicyRule are to be performed if and only if the PolicyRule condition (whether it is represented in DNF or CNF) evaluates to TRUE.

The conditions and actions associated with a policy rule are modeled, respectively, with subclasses of the classes PolicyCondition and PolicyAction. These condition and action objects are tied to instances of PolicyRule by the PolicyConditionInPolicyRule and PolicyActionInPolicyRule aggregations.

As illustrated above in Section 3, a policy rule may also be associated with one or more policy time periods, indicating the schedule according to which the policy rule is active and inactive. In this case it is the PolicyRuleValidityPeriod aggregation that provides the linkage.

A policy rule is illustrated conceptually in Figure 6. below.

```
+------------------------------------------------+
|                    PolicyRule                  |
|                                                |
| +--------------------+     +------------------+ |
| | PolicyCondition(s) |     | PolicyAction(s) | |
| +--------------------+     +------------------+ |
|                                                |
|        +------------------------------+        |
|        | PolicyTimePeriodCondition(s) |        |
|        +------------------------------+        |
+------------------------------------------------+
```

Figure 6. Overview of the PolicyRule Class

The PolicyRule class uses the property ConditionListType, to indicate whether the conditions for the rule are in DNF or CNF. The PolicyConditionInPolicyRule aggregation contains two additional properties to complete the representation of the rule's conditional expression. The first of these properties is an integer to partition the referenced conditions into one or more groups, and the second is a Boolean to indicate whether a referenced condition is negated. An example shows how ConditionListType and these two additional properties provide a unique representation of a set of conditions in either DNF or CNF.

Suppose we have a PolicyRule that aggregates five PolicyConditions C1 through C5, with the following values in the properties of the five PolicyConditionInPolicyRule associations:

```
   C1:  GroupNumber = 1, ConditionNegated = FALSE
   C2:  GroupNumber = 1, ConditionNegated = TRUE
   C3:  GroupNumber = 1, ConditionNegated = FALSE
   C4:  GroupNumber = 2, ConditionNegated = FALSE
   C5:  GroupNumber = 2, ConditionNegated = FALSE
```

If ConditionListType = DNF, then the overall condition for the PolicyRule is:

```
   (C1 AND (NOT C2) AND C3) OR (C4 AND C5)
```

On the other hand, if ConditionListType = CNF, then the overall condition for the PolicyRule is:

```
   (C1 OR (NOT C2) OR C3) AND (C4 OR C5)
```

In both cases, there is an unambiguous specification of the overall condition that is tested to determine whether to perform the actions associated with the PolicyRule.

The class definition is as follows:

NAME	PolicyRule
DESCRIPTION	The central class for representing the "If Condition then Action" semantics associated with a policy rule.
DERIVED FROM	Policy
ABSTRACT	FALSE
PROPERTIES	Enabled ConditionListType RuleUsage Priority Mandatory SequencedActions PolicyRoles

The PolicyRule class is directly instantiable. In an implementation, various key/identification properties MUST be defined. The keys for a native CIM implementation are defined in Appendix A, Section 13.1.2. Keys for an LDAP implementation will be defined in the LDAP mapping of this information model [11].

6.3.1. The Property "Enabled"

This property indicates whether a policy rule is currently enabled, from an administrative point of view. Its purpose is to allow a policy administrator to enable or disable a policy rule without having to add it to, or remove it from, the policy repository.

The property also supports the value 'enabledForDebug'. When the property has this value, the entity evaluating the policy condition(s) is being told to evaluate the conditions for the policy rule, but not to perform the actions if the conditions evaluate to TRUE. This value serves as a debug vehicle when attempting to determine what policies would execute in a particular scenario, without taking any actions to change state during the debugging.

The property definition is as follows:

NAME	Enabled
DESCRIPTION	An enumeration indicating whether a policy rule is administratively enabled, administratively disabled, or enabled for debug mode.
SYNTAX	uint16
VALUES	enabled(1), disabled(2), enabledForDebug(3)
DEFAULT VALUE	enabled(1)

6.3.2. The Property "ConditionListType"

This property is used to specify whether the list of policy conditions associated with this policy rule is in disjunctive normal form (DNF) or conjunctive normal form (CNF). If this property is not present, the list type defaults to DNF. The property definition is as follows:

NAME	ConditionListType
DESCRIPTION	Indicates whether the list of policy conditions associated with this policy rule is in disjunctive normal form (DNF) or conjunctive normal form (CNF).
SYNTAX	uint16
VALUES	DNF(1), CNF(2)
DEFAULT VALUE	DNF(1)

6.3.3. The Property "RuleUsage"

This property is a free-form string that recommends how this policy should be used. The property definition is as follows:

NAME	RuleUsage
DESCRIPTION	This property is used to provide guidelines on how this policy should be used.
SYNTAX	string

6.3.4. The Property "Priority"

This property provides a non-negative integer for prioritizing policy rules relative to each other. Larger integer values indicate higher priority. Since one purpose of this property is to allow specific, ad hoc policy rules to temporarily override established policy rules, an instance that has this property set has a higher priority than all instances that use or set the default value of zero.

Prioritization among policy rules provides a basic mechanism for resolving policy conflicts.

The property definition is as follows:

NAME	Priority
DESCRIPTION	A non-negative integer for prioritizing this PolicyRule relative to other PolicyRules. A larger value indicates a higher priority.
SYNTAX	uint16
DEFAULT VALUE	0

6.3.5. The Property "Mandatory"

This property indicates whether evaluation (and possibly action execution) of a PolicyRule is mandatory or not. Its concept is similar to the ability to mark packets for delivery or possible discard, based on network traffic and device load.

The evaluation of a PolicyRule MUST be attempted if the Mandatory property value is TRUE. If the Mandatory property value of a PolicyRule is FALSE, then the evaluation of the rule is "best effort" and MAY be ignored.

The property definition is as follows:

NAME	Mandatory
DESCRIPTION	A flag indicating that the evaluation of the PolicyConditions and execution of PolicyActions (if the condition list evaluates to TRUE) is required.
SYNTAX	boolean
DEFAULT VALUE	TRUE

6.3.6. The Property "SequencedActions"

This property gives a policy administrator a way of specifying how the ordering of the policy actions associated with this PolicyRule is to be interpreted. Three values are supported:

- o mandatory(1): Do the actions in the indicated order, or don't do them at all.
- o recommended(2): Do the actions in the indicated order if you can, but if you can't do them in this order, do them in another order if you can.
- o dontCare(3): Do them -- I don't care about the order.

When error / event reporting is addressed for the Policy Framework, suitable codes will be defined for reporting that a set of actions could not be performed in an order specified as mandatory (and thus were not performed at all), that a set of actions could not be performed in a recommended order (and moreover could not be performed in any order), or that a set of actions could not be performed in a recommended order (but were performed in a different order). The property definition is as follows:

```
NAME            SequencedActions
DESCRIPTION     An enumeration indicating how to interpret the
                action ordering indicated via the
                PolicyActionInPolicyRule aggregation.
SYNTAX          uint16
VALUES          mandatory(1), recommended(2), dontCare(3)
DEFAULT VALUE   dontCare(3)
```

6.3.7. The Multi-valued Property "PolicyRoles"

This property represents the roles and role combinations associated with a policy rule. Each value represents one role combination. Since this is a multi-valued property, more than one role combination can be associated with a single policy rule. Each value is a string of the form

```
<RoleName>[&&<RoleName>]*
```

where the individual role names appear in alphabetical order (according to the collating sequence for UCS-2). The property definition is as follows:

NAME	PolicyRoles
DESCRIPTION	A set of strings representing the roles and role combinations associated with a policy rule. Each value represents one role combination.
SYNTAX	string

6.4. The Abstract Class "PolicyCondition"

The purpose of a policy condition is to determine whether or not the set of actions (aggregated in the PolicyRule that the condition applies to) should be executed or not. For the purposes of the Policy Core Information Model, all that matters about an individual PolicyCondition is that it evaluates to TRUE or FALSE. (The individual PolicyConditions associated with a PolicyRule are combined to form a compound expression in either DNF or CNF, but this is accomplished via the ConditionListType property, discussed above, and by the properties of the PolicyConditionInPolicyRule aggregation, introduced above and discussed further in Section 7.6 below.) A logical structure within an individual PolicyCondition may also be introduced, but this would have to be done in a subclass of PolicyCondition.

Because it is general, the PolicyCondition class does not itself contain any "real" conditions. These will be represented by properties of the domain-specific subclasses of PolicyCondition.

```
+---------------------------------------------------------------+
|                  Policy Conditions in DNF                     |
| +-------------------------+         +-----------------------+ |
| |       AND list          |         |      AND list         | |
| |  +-------------------+  |         |  +-----------------+  | |
| |  |  PolicyCondition  |  |         |  | PolicyCondition |  | |
| |  +-------------------+  |         |  +-----------------+  | |
| |  +-------------------+  |         |  +-----------------+  | |
| |  |  PolicyCondition  |  |   ...   |  | PolicyCondition |  | |
| |  +-------------------+  |  ORed   |  +-----------------+  | |
| |          ...            |         |         ...           | |
| |         ANDed           |         |        ANDed          | |
| |  +-------------------+  |         |  +-----------------+  | |
| |  |  PolicyCondition  |  |         |  | PolicyCondition |  | |
| |  +-------------------+  |         |  +-----------------+  | |
| +-------------------------+         +-----------------------+ |
+---------------------------------------------------------------+
```

Figure 7. Overview of Policy Conditions in DNF

This figure illustrates that when policy conditions are in DNF, there are one or more sets of conditions that are ANDed together to form AND lists. An AND list evaluates to TRUE if and only if all of its constituent conditions evaluate to TRUE. The overall condition then evaluates to TRUE if and only if at least one of its constituent AND lists evaluates to TRUE.

```
+---------------------------------------------------------------+
|                    Policy Conditions in CNF                   |
| +-------------------------+       +-----------------------+   |
| |        OR list          |       |       OR list         |   |
| |  +-------------------+  |       | +-----------------+   |   |
| |  | PolicyCondition   |  |       | | PolicyCondition |   |   |
| |  +-------------------+  |       | +-----------------+   |   |
| |  +-------------------+  |       | +-----------------+   |   |
| |  | PolicyCondition   |  |  ...  | | PolicyCondition |   |   |
| |  +-------------------+  | ANDed | +-----------------+   |   |
| |          ...            |       |         ...           |   |
| |         ORed            |       |        ORed           |   |
| |  +-------------------+  |       | +-----------------+   |   |
| |  | PolicyCondition   |  |       | | PolicyCondition |   |   |
| |  +-------------------+  |       | +-----------------+   |   |
| +-------------------------+       +-----------------------+   |
+---------------------------------------------------------------+
```

Figure 8. Overview of Policy Conditions in CNF

In this figure, the policy conditions are in CNF. Consequently, there are one or more OR lists, each of which evaluates to TRUE if and only if at least one of its constituent conditions evaluates to TRUE. The overall condition then evaluates to TRUE if and only if ALL of its constituent OR lists evaluate to TRUE.

The class definition of PolicyCondition is as follows:

NAME	PolicyCondition
DESCRIPTION	A class representing a rule-specific or reusable policy condition to be evaluated in conjunction with a policy rule.
DERIVED FROM	Policy
ABSTRACT	TRUE
PROPERTIES	NONE

No properties are defined for this class since it inherits all its properties from Policy. The class exists as an abstract superclass for domain-specific policy conditions, defined in subclasses. In an implementation, various key/identification properties MUST be defined for the class or its instantiable subclasses. The keys for a native

CIM implementation are defined in Appendix A, Section 13.2. Keys for an LDAP implementation will be defined in the LDAP mapping of this information model [11].

When identifying and using the PolicyCondition class, it is necessary to remember that a condition can be rule-specific or reusable. This was discussed above in Section 5.1. The distinction between the two types of policy conditions lies in the associations in which an instance can participate, and in how the different instances are named. Conceptually, a reusable policy condition resides in a policy repository, and is named within the scope of that repository. On the other hand, a rule-specific policy condition is, as the name suggests, named within the scope of the single policy rule to which it is related.

The distinction between rule-specific and reusable PolicyConditions affects the CIM naming, defined in Appendix A, and the LDAP mapping [11].

6.5. The Class "PolicyTimePeriodCondition"

This class provides a means of representing the time periods during which a policy rule is valid, i.e., active. At all times that fall outside these time periods, the policy rule has no effect. A policy rule is treated as valid at all times if it does not specify a PolicyTimePeriodCondition.

In some cases a PDP may need to perform certain setup / cleanup actions when a policy rule becomes active / inactive. For example, sessions that were established while a policy rule was active might need to be taken down when the rule becomes inactive. In other cases, however, such sessions might be left up: in this case, the effect of deactivating the policy rule would just be to prevent the establishment of new sessions. Setup / cleanup behaviors on validity period transitions are not currently addressed by the PCIM, and must be specified in 'guideline' documents, or via subclasses of PolicyRule, PolicyTimePeriodCondition or other concrete subclasses of Policy. If such behaviors need to be under the control of the policy administrator, then a mechanism to allow this control must also be specified in the subclass.

PolicyTimePeriodCondition is defined as a subclass of PolicyCondition. This is to allow the inclusion of time-based criteria in the AND/OR condition definitions for a PolicyRule.

Instances of this class may have up to five properties identifying time periods at different levels. The values of all the properties present in an instance are ANDed together to determine the validity

```
   period(s) for the instance.  For example, an instance with an overall
   validity range of January 1, 2000 through December 31, 2000; a month
   mask that selects March and April; a day-of-the-week mask that
   selects Fridays; and a time of day range of 0800 through 1600 would
   represent the following time periods:

      Friday, March  5, 2000, from 0800 through 1600;
      Friday, March 12, 2000, from 0800 through 1600;
      Friday, March 19, 2000, from 0800 through 1600;
      Friday, March 26, 2000, from 0800 through 1600;
      Friday, April  2, 2000, from 0800 through 1600;
      Friday, April  9, 2000, from 0800 through 1600;
      Friday, April 16, 2000, from 0800 through 1600;
      Friday, April 23, 2000, from 0800 through 1600;
      Friday, April 30, 2000, from 0800 through 1600.

   Properties not present in an instance of PolicyTimePeriodCondition
   are implicitly treated as having their value "always enabled".  Thus,
   in the example above, the day-of-the-month mask is not present, and
   so the validity period for the instance implicitly includes a day-
   of-the-month mask that selects all days of the month.  If we apply
   this "missing property" rule to its fullest, we see that there is a
   second way to indicate that a policy rule is always enabled: have it
   point to an instance of PolicyTimePeriodCondition whose only
   properties are its naming properties.

   The property LocalOrUtcTime indicates whether the times represented
   in the other five time-related properties of an instance of
   PolicyTimePeriodCondition are to be interpreted as local times for
   the location where a policy rule is being applied, or as UTC times.

   The class definition is as follows.

   NAME             PolicyTimePeriodCondition
   DESCRIPTION      A class that provides the capability of enabling /
                    disabling a policy rule according to a
                    pre-determined schedule.
   DERIVED FROM     PolicyCondition
   ABSTRACT         FALSE
   PROPERTIES       TimePeriod
                    MonthOfYearMask
                    DayOfMonthMask
                    DayOfWeekMask
                    TimeOfDayMask
                    LocalOrUtcTime
```

6.5.1. The Property "TimePeriod"

This property identifies an overall range of calendar dates and times over which a policy rule is valid. It reuses the format for an explicit time period defined in RFC 2445 (reference [10]): a string representing a starting date and time, in which the character 'T' indicates the beginning of the time portion, followed by the solidus character '/', followed by a similar string representing an end date and time. The first date indicates the beginning of the range, while the second date indicates the end. Thus, the second date and time must be later than the first. Date/times are expressed as substrings of the form "yyyymmddThhmmss". For example:

```
   20000101T080000/20000131T120000

      January 1, 2000, 0800 through January 31, 2000, noon
```

There are also two special cases in which one of the date/time strings is replaced with a special string defined in RFC 2445.

- o If the first date/time is replaced with the string "THISANDPRIOR", then the property indicates that a policy rule is valid [from now] until the date/time that appears after the '/'.

- o If the second date/time is replaced with the string "THISANDFUTURE", then the property indicates that a policy rule becomes valid on the date/time that appears before the '/', and remains valid from that point on.

Note that RFC 2445 does not use these two strings in connection with explicit time periods. Thus the PCIM is combining two elements from RFC 2445 that are not combined in the RFC itself.

The property definition is as follows:

NAME	TimePeriod
DESCRIPTION	The range of calendar dates on which a policy rule is valid.
SYNTAX	string
FORMAT	yyyymmddThhmmss/yyyymmddThhmmss, where the first date/time may be replaced with the string "THISANDPRIOR" or the second date/time may be replaced with the string "THISANDFUTURE"

6.5.2. The Property "MonthOfYearMask"

The purpose of this property is to refine the definition of the valid time period that is defined by the TimePeriod property, by explicitly specifying the months when the policy is valid. These properties work together, with the TimePeriod used to specify the overall time period during which the policy might be valid, and the MonthOfYearMask used to pick out the specific months within that time period when the policy is valid.

This property is formatted as an octet string of size 2, consisting of 12 bits identifying the 12 months of the year, beginning with January and ending with December, followed by 4 bits that are always set to '0'. For each month, the value '1' indicates that the policy is valid for that month, and the value '0' indicates that it is not valid. The value X'08 30', for example, indicates that a policy rule is valid only in the months May, November, and December.

See section 5.4 for details of how CIM represents a single-valued octet string property such as this one. (Basically, CIM prepends a 4-octet length to the octet string.)

If this property is omitted, then the policy rule is treated as valid for all twelve months. The property definition is as follows:

NAME	MonthOfYearMask
DESCRIPTION	A mask identifying the months of the year in which a policy rule is valid.
SYNTAX	octet string
FORMAT	X'hh h0'

6.5.3. The Property "DayOfMonthMask"

The purpose of this property is to refine the definition of the valid time period that is defined by the TimePeriod property, by explicitly specifying the days of the month when the policy is valid. These properties work together, with the TimePeriod used to specify the overall time period during which the policy might be valid, and the DayOfMonthMask used to pick out the specific days of the month within that time period when the policy is valid.

This property is formatted as an octet string of size 8, consisting of 31 bits identifying the days of the month counting from the beginning, followed by 31 more bits identifying the days of the month counting from the end, followed by 2 bits that are always set to '0'. For each day, the value '1' indicates that the policy is valid for that day, and the value '0' indicates that it is not valid.

The value X'80 00 00 01 00 00 00 00', for example, indicates that a policy rule is valid on the first and last days of the month.

For months with fewer than 31 days, the digits corresponding to days that the months do not have (counting in both directions) are ignored.

The encoding of the 62 significant bits in the octet string matches that used for the schedDay object in the DISMAN-SCHEDULE-MIB. See reference [8] for more details on this object.

See section 5.4 for details of how CIM represents a single-valued octet string property such as this one. (Basically, CIM prepends a 4-octet length to the octet string.)

The property definition is as follows:

```
   NAME             DayOfMonthMask
   DESCRIPTION      A mask identifying the days of the month on
                    which a policy rule is valid.
   SYNTAX           octet string
   FORMAT           X'hh hh hh hh hh hh hh hh'
```

6.5.4. The Property "DayOfWeekMask"

The purpose of this property is to refine the definition of the valid time period that is defined by the TimePeriod property by explicitly specifying the days of the week when the policy is valid. These properties work together, with the TimePeriod used to specify the overall time period when the policy might be valid, and the DayOfWeekMask used to pick out the specific days of the week in that time period when the policy is valid.

This property is formatted as an octet string of size 1, consisting of 7 bits identifying the 7 days of the week, beginning with Sunday and ending with Saturday, followed by 1 bit that is always set to '0'. For each day of the week, the value '1' indicates that the policy is valid for that day, and the value '0' indicates that it is not valid.

The value X'7C', for example, indicates that a policy rule is valid Monday through Friday.

See section 5.4 for details of how CIM represents a single-valued octet string property such as this one. (Basically, CIM prepends a 4-octet length to the octet string.)

The property definition is as follows:

```
NAME              DayOfWeekMask
DESCRIPTION       A mask identifying the days of the week on which
                  a policy rule is valid.
SYNTAX            octet string
FORMAT            B'bbbb bbb0'
```

6.5.5. The Property "TimeOfDayMask"

The purpose of this property is to refine the definition of the valid time period that is defined by the TimePeriod property by explicitly specifying a range of times in a day the policy is valid for. These properties work together, with the TimePeriod used to specify the overall time period that the policy is valid for, and the TimeOfDayMask used to pick out which range of time periods in a given day of that time period the policy is valid for.

This property is formatted in the style of RFC 2445 [10]: a time string beginning with the character 'T', followed by the solidus character '/', followed by a second time string. The first time indicates the beginning of the range, while the second time indicates the end. Times are expressed as substrings of the form "Thhmmss".

The second substring always identifies a later time than the first substring. To allow for ranges that span midnight, however, the value of the second string may be smaller than the value of the first substring. Thus, "T080000/T210000" identifies the range from 0800 until 2100, while "T210000/T080000" identifies the range from 2100 until 0800 of the following day.

When a range spans midnight, it by definition includes parts of two successive days. When one of these days is also selected by either the MonthOfYearMask, DayOfMonthMask, and/or DayOfWeekMask, but the other day is not, then the policy is active only during the portion of the range that falls on the selected day. For example, if the range extends from 2100 until 0800, and the day of week mask selects Monday and Tuesday, then the policy is active during the following three intervals:

From midnight Sunday until 0800 Monday;
From 2100 Monday until 0800 Tuesday;
From 2100 Tuesday until 23:59:59 Tuesday.

The property definition is as follows:

```
NAME            TimeOfDayMask
DESCRIPTION     The range of times at which a policy rule is
                valid.  If the second time is earlier than the
                first, then the interval spans midnight.
SYNTAX          string
FORMAT          Thhmmss/Thhmmss
```

6.5.6. The Property "LocalOrUtcTime"

This property indicates whether the times represented in the TimePeriod property and in the various Mask properties represent local times or UTC times. There is no provision for mixing of local times and UTC times: the value of this property applies to all of the other time-related properties.

The property definition is as follows:

```
NAME            LocalOrUtcTime
DESCRIPTION     An indication of whether the other times in this
                instance represent local times or UTC times.
SYNTAX          uint16
VALUES          localTime(1), utcTime(2)
DEFAULT VALUE   utcTime(2)
```

6.6. The Class "VendorPolicyCondition"

The purpose of this class is to provide a general extension mechanism for representing policy conditions that have not been modeled with specific properties. Instead, the two properties Constraint and ConstraintEncoding are used to define the content and format of the condition, as explained below.

As its name suggests, this class is intended for vendor-specific extensions to the Policy Core Information Model. Standardized extensions are not expected to use this class.

The class definition is as follows:

```
NAME            VendorPolicyCondition
DESCRIPTION     A class that defines a registered means to
                describe a policy condition.
DERIVED FROM    PolicyCondition
ABSTRACT        FALSE
PROPERTIES      Constraint[ ]
                ConstraintEncoding
```

6.6.1. The Multi-valued Property "Constraint"

This property provides a general extension mechanism for representing policy conditions that have not been modeled with specific properties. The format of the octet strings in the array is left unspecified in this definition. It is determined by the OID value stored in the property ConstraintEncoding. Since ConstraintEncoding is single-valued, all the values of Constraint share the same format and semantics.

See Section 5.4 for a description of how CIM encodes an array of octet strings like this one.

A policy decision point can readily determine whether it supports the values stored in an instance of Constraint by checking the OID value from ConstraintEncoding against the set of OIDs it recognizes. The action for the policy decision point to take in case it does not recognize the format of this data could itself be modeled as a policy rule, governing the behavior of the policy decision point.

The property is defined as follows:

NAME	Constraint
DESCRIPTION	Extension mechanism for representing constraints that have not been modeled as specific properties. The format of the values is identified by the OID stored in the property ConstraintEncoding.
SYNTAX	octet string

6.6.2. The Property "ConstraintEncoding"

This property identifies the encoding and semantics of the Constraint property values in this instance. The value of this property is a single string, representing a single OID.

The property is defined as follows:

NAME	ConstraintEncoding
DESCRIPTION	An OID encoded as a string, identifying the format and semantics for this instance's Constraint property. The value is a dotted sequence of decimal digits (for example, "1.2.100.200") representing the arcs of the OID. The characters in the string are the UCS-2 characters corresponding to the US ASCII encodings of the numeric characters and the period.
SYNTAX	string

6.7. The Abstract Class "PolicyAction"

The purpose of a policy action is to execute one or more operations that will affect network traffic and/or systems, devices, etc., in order to achieve a desired state. This (new) state provides one or more (new) behaviors. A policy action ordinarily changes the configuration of one or more elements.

A PolicyRule contains one or more policy actions. A policy administrator can assign an order to the actions associated with a PolicyRule, complete with an indication of whether the indicated order is mandatory, recommended, or of no significance. Ordering of the actions associated with a PolicyRule is accomplished via a property in the PolicyActionInPolicyRule aggregation.

The actions associated with a PolicyRule are executed if and only if the overall condition(s) of the PolicyRule evaluates to TRUE.

The class definition of PolicyAction is as follows:

NAME	PolicyAction
DESCRIPTION	A class representing a rule-specific or reusable policy action to be performed if the condition for a policy rule evaluates to TRUE.
DERIVED FROM	Policy
ABSTRACT	TRUE
PROPERTIES	NONE

No properties are defined for this class since it inherits all its properties from Policy. The class exists as an abstract superclass for domain-specific policy actions, defined in subclasses. In an implementation, various key/identification properties MUST be defined for the class or its instantiable subclasses. The keys for a native CIM implementation are defined in Appendix A, Section 13.3. Keys for an LDAP implementation will be defined in the LDAP mapping of this information model [11].

When identifying and using the PolicyAction class, it is necessary to remember that an action can be rule-specific or reusable. This was discussed above in Section 5.1. The distinction between the two types of policy actions lies in the associations in which an instance can participate, and in how the different instances are named. Conceptually, a reusable policy action resides in a policy repository, and is named within the scope of that repository. On the other hand, a rule-specific policy action is named within the scope of the single policy rule to which it is related.

The distinction between rule-specific and reusable PolicyActions affects the CIM naming, defined in Appendix A, and the LDAP mapping [11].

6.8. The Class "VendorPolicyAction"

The purpose of this class is to provide a general extension mechanism for representing policy actions that have not been modeled with specific properties. Instead, the two properties ActionData and ActionEncoding are used to define the content and format of the action, as explained below.

As its name suggests, this class is intended for vendor-specific extensions to the Policy Core Information Model. Standardized extensions are not expected to use this class.

The class definition is as follows:

```
NAME              VendorPolicyAction
DESCRIPTION       A class that defines a registered means to
                  describe a policy action.
DERIVED FROM      PolicyAction
ABSTRACT          FALSE
PROPERTIES        ActionData[ ]
                  ActionEncoding
```

6.8.1. The Multi-valued Property "ActionData"

This property provides a general extension mechanism for representing policy actions that have not been modeled with specific properties. The format of the octet strings in the array is left unspecified in this definition. It is determined by the OID value stored in the property ActionEncoding. Since ActionEncoding is single-valued, all the values of ActionData share the same format and semantics. See Section 5.4 for a discussion of how CIM encodes an array of octet strings like this one.

A policy decision point can readily determine whether it supports the values stored in an instance of ActionData by checking the OID value from ActionEncoding against the set of OIDs it recognizes. The action for the policy decision point to take in case it does not recognize the format of this data could itself be modeled as a policy rule, governing the behavior of the policy decision point.

The property is defined as follows:

```
NAME            ActionData
DESCRIPTION     Extension mechanism for representing actions that
                have not been modeled as specific properties.  The
                format of the values is identified by the OID
                stored in the property ActionEncoding.
SYNTAX          octet string
```

6.8.2. The Property "ActionEncoding"

This property identifies the encoding and semantics of the ActionData property values in this instance. The value of this property is a single string, representing a single OID.

The property is defined as follows:

```
NAME            ActionEncoding
DESCRIPTION     An OID encoded as a string, identifying the format
                and semantics for this instance's ActionData
                property.  The value is a dotted sequence of
                decimal digits (for example, "1.2.100.200")
                representing the arcs of the OID.  The characters
                in the string are the UCS-2 characters
                corresponding to the US ASCII encodings of the
                numeric characters and the period.
SYNTAX          string
```

6.9. The Class "PolicyRepository"

The class definition of PolicyRepository is as follows:

```
NAME            PolicyRepository
DESCRIPTION     A class representing an administratively defined
                container for reusable policy-related
                information.  This class does not introduce any
                additional properties beyond those in its
                superclass AdminDomain.  It does, however,
                participate in a number of unique associations.
DERIVED FROM    AdminDomain
ABSTRACT        FALSE
```

7. Association and Aggregation Definitions

The first two subsections of this section introduce associations and aggregations as they are used in CIM. The remaining subsections present the class definitions for the associations and aggregations that are part of the Policy Core Information Model.

7.1. Associations

An association is a CIM construct representing a relationship between two (or theoretically more) objects. It is modeled as a class containing typically two object references. Associations can be defined between classes without affecting any of the related classes. That is, addition of an association does not affect the interface of the related classes.

7.2. Aggregations

An aggregation is a strong form of an association, which usually represents a "whole-part" or a "collection" relationship. For example, CIM uses an aggregation to represent the containment relationship between a system and the components that make up the system. Aggregation as a "whole-part" relationship often implies, but does not require, that the aggregated objects have mutual dependencies.

7.3. The Abstract Aggregation "PolicyComponent

This abstract aggregation defines two object references that will be overridden in each of five subclasses, to become references to the concrete policy classes PolicyGroup, PolicyRule, PolicyCondition, PolicyAction, and PolicyTimePeriodCondition. The value of the abstract superclass is to convey that all five subclasses have the same "whole- part" semantics, and for ease of query to locate all "components" of a PolicyGroup or PolicyRule.

The class definition for the aggregation is as follows:

```
  NAME           PolicyComponent
  DESCRIPTION    A generic aggregation used to establish 'part of'
                 relationships between the subclasses of
                 Policy.  For example, the
                 PolicyConditionInPolicyRule aggregation defines
                 that PolicyConditions are part of a PolicyRule.
  ABSTRACT       TRUE
  PROPERTIES     GroupComponent[ref Policy[0..n]]
                 PartComponent[ref Policy[0..n]]
```

7.4. The Aggregation "PolicyGroupInPolicyGroup"

The PolicyGroupInPolicyGroup aggregation enables policy groups to be nested. This is critical for scalability and manageability, as it enables complex policies to be constructed from multiple simpler policies for administrative convenience. For example, a policy group representing policies for the US might have nested within it policy groups for the Eastern and Western US.

A PolicyGroup may aggregate other PolicyGroups via this aggregation, or it may aggregate PolicyRules via the PolicyRuleInPolicyGroup aggregation. Note that it is assumed that this aggregation is used to form directed acyclic graphs and NOT ring structures.The class definition for the aggregation is as follows:

```
NAME             PolicyGroupInPolicyGroup
DESCRIPTION      A class representing the aggregation of
                 PolicyGroups by a higher-level PolicyGroup.
DERIVED FROM     PolicyComponent
ABSTRACT         FALSE
PROPERTIES       GroupComponent[ref PolicyGroup[0..n]]
                 PartComponent[ref PolicyGroup[0..n]]
```

7.4.1. The Reference "GroupComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyGroup that contains one or more other PolicyGroups. Note that for any single instance of the aggregation class PolicyGroupInPolicyGroup, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more than one PolicyGroups that contain any given PolicyGroup.

7.4.2. The Reference "PartComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyGroup contained by one or more other PolicyGroups. Note that for any single instance of the aggregation class PolicyGroupInPolicyGroup, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given PolicyGroup may contain 0, 1, or more than one other PolicyGroups.

7.5. The Aggregation "PolicyRuleInPolicyGroup"

A policy group may aggregate one or more policy rules, via the PolicyRuleInPolicyGroup aggregation. Grouping of policy rules into a policy group is again for administrative convenience; a policy rule may also be used by itself, without belonging to a policy group.

A PolicyGroup may aggregate PolicyRules via this aggregation, or it may aggregate other PolicyGroups via the PolicyGroupInPolicyGroup aggregation.

The class definition for the aggregation is as follows:

```
NAME            PolicyRuleInPolicyGroup
DESCRIPTION     A class representing the aggregation of
                PolicyRules by a PolicyGroup.
DERIVED FROM    PolicyComponent
ABSTRACT        FALSE
PROPERTIES      GroupComponent[ref PolicyGroup[0..n]]
                PartComponent[ref PolicyRule[0..n]]
```

7.5.1. The Reference "GroupComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyGroup that contains one or more PolicyRules. Note that for any single instance of the aggregation class PolicyRuleInPolicyGroup, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more than one PolicyGroups that contain any given PolicyRule.

7.5.2. The Reference "PartComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyRule contained by one or more PolicyGroups. Note that for any single instance of the aggregation class PolicyRuleInPolicyGroup, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given PolicyGroup may contain 0, 1, or more than one PolicyRules.

7.6. The Aggregation "PolicyConditionInPolicyRule"

A policy rule aggregates zero or more instances of the PolicyCondition class, via the PolicyConditionInPolicyRule association. A policy rule that aggregates zero policy conditions must indicate in its class definition what "triggers" the performance of its actions. In short, it must describe its implicit PolicyConditions, since none are explicitly associated. For example, there might be a subclass of PolicyRule named "HttpPolicyRule", where the class definition assumes that the condition, "If HTTP traffic," is true before the rule's actions would be performed. There is no need to formalize and instantiate this condition, since it is obvious in the semantics of the PolicyRule.

The conditions aggregated by a policy rule are grouped into two levels of lists: either an ORed set of ANDed sets of conditions (DNF, the default) or an ANDed set of ORed sets of conditions (CNF). Individual conditions in these lists may be negated. The property ConditionListType (in PolicyRule) specifies which of these two grouping schemes applies to a particular PolicyRule. The conditions are used to determine whether to perform the actions associated with the PolicyRule.

One or more policy time periods may be among the conditions associated with a policy rule via the PolicyConditionInPolicyRule association. In this case, the time periods are simply additional conditions to be evaluated along with any other conditions specified for the rule.

The class definition for the aggregation is as follows:

```
NAME            PolicyConditionInPolicyRule
DESCRIPTION     A class representing the aggregation of
                PolicyConditions by a PolicyRule.
DERIVED FROM    PolicyComponent
ABSTRACT        FALSE
PROPERTIES      GroupComponent[ref PolicyRule[0..n]]
                PartComponent[ref PolicyCondition[0..n]]
                GroupNumber
                ConditionNegated
```

7.6.1. The Reference "GroupComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyRule that contains one or more PolicyConditions. Note that for any single instance of the aggregation class PolicyConditionInPolicyRule, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more than one PolicyRules that contain any given PolicyCondition.

7.6.2. The Reference "PartComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyCondition contained by one or more PolicyRules. Note that for any single instance of the aggregation class PolicyConditionInPolicyRule, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given PolicyRule may contain 0, 1, or more than one PolicyConditions.

7.6.3. The Property "GroupNumber"

This property contains an integer identifying the group to which the condition referenced by the PartComponent property is assigned in forming the overall conditional expression for the policy rule identified by the GroupComponent reference.

The property is defined as follows:

```
NAME            GroupNumber
DESCRIPTION     Unsigned integer indicating the group to which
                the condition identified by the PartComponent
                property is to be assigned.
SYNTAX          uint16
DEFAULT         0
```

7.6.4. The Property "ConditionNegated"

This property is a boolean, indicating whether the condition referenced by the PartComponent property is negated in forming the overall conditional expression for the policy rule identified by the GroupComponent reference.

The property is defined as follows:

```
NAME            ConditionNegated
DESCRIPTION     Indication of whether the condition identified by
                the PartComponent property is negated.  (TRUE
                indicates that the condition is negated, FALSE
                indicates that it is not negated.)
SYNTAX          boolean
DEFAULT         FALSE
```

7.7. The Aggregation "PolicyRuleValidityPeriod"

A different relationship between a policy rule and a policy time period (than PolicyConditionInPolicyRule) is represented by the PolicyRuleValidityPeriod aggregation. The latter describes scheduled activation and deactivation of the policy rule.

If a policy rule is associated with multiple policy time periods via this association, then the rule is active if at least one of the time periods indicates that it is active. (In other words, the time periods are ORed to determine whether the rule is active.) A policy time period may be aggregated by multiple policy rules. A rule that does not point to a policy time period via this aggregation is, from the point of view of scheduling, always active. It may, however, be inactive for other reasons.

Time periods are a general concept that can be used in other applications. However, they are mentioned explicitly here in this specification since they are frequently used in policy applications.

The class definition for the aggregation is as follows:

```
NAME            PolicyRuleValidityPeriod
DESCRIPTION     A class representing the aggregation of
                PolicyTimePeriodConditions by a PolicyRule.
DERIVED FROM    PolicyComponent
ABSTRACT        FALSE
PROPERTIES      GroupComponent[ref PolicyRule[0..n]]
                PartComponent[ref PolicyTimePeriodCondition[0..n]]
```

7.7.1. The Reference "GroupComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyRule that contains one or more PolicyTimePeriodConditions. Note that for any single instance of the aggregation class PolicyRuleValidityPeriod, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more than one PolicyRules that contain any given PolicyTimePeriodCondition.

7.7.2. The Reference "PartComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyTimePeriodCondition contained by one or more PolicyRules. Note that for any single instance of the aggregation class PolicyRuleValidityPeriod, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given PolicyRule may contain 0, 1, or more than one PolicyTimePeriodConditions.

7.8. The Aggregation "PolicyActionInPolicyRule"

A policy rule may aggregate zero or more policy actions. A policy rule that aggregates zero policy actions must indicate in its class definition what actions are taken when the rule's conditions evaluate to TRUE. In short, it must describe its implicit PolicyActions, since none are explicitly associated. For example, there might be a subclass of PolicyRule representing a Diffserv absolute dropper, where the subclass itself indicates the action to be taken. There is no need to formalize and instantiate this action, since it is obvious in the semantics of the PolicyRule.

The actions associated with a PolicyRule may be given a required order, a recommended order, or no order at all. For actions represented as separate objects, the PolicyActionInPolicyRule aggregation can be used to express an order.

This aggregation does not indicate whether a specified action order is required, recommended, or of no significance; the property SequencedActions in the aggregating instance of PolicyRule provides this indication.

The class definition for the aggregation is as follows:

```
NAME            PolicyActionInPolicyRule
DESCRIPTION     A class representing the aggregation of
                PolicyActions by a PolicyCondition.
DERIVED FROM    PolicyComponent
ABSTRACT        FALSE
PROPERTIES      GroupComponent[ref PolicyRule[0..n]]
                PartComponent[ref PolicyAction[0..n]]
                ActionOrder
```

7.8.1. The Reference "GroupComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyRule that contains one or more PolicyActions. Note that for any single instance of the aggregation class PolicyActionInPolicyRule, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more than one PolicyRules that contain any given PolicyAction.

7.8.2. The Reference "PartComponent"

This property is inherited from PolicyComponent, and overridden to become an object reference to a PolicyAction contained by one or more PolicyRules. Note that for any single instance of the aggregation class PolicyActionInPolicyRule, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given PolicyRule may contain 0, 1, or more than one PolicyActions.

7.8.3. The Property "ActionOrder"

This property provides an unsigned integer 'n' that indicates the relative position of an action in the sequence of actions associated with a policy rule. When 'n' is a positive integer, it indicates a place in the sequence of actions to be performed, with smaller integers indicating earlier positions in the sequence. The special value '0' indicates "don't care". If two or more actions have the same non-zero sequence number, they may be performed in any order, but they must all be performed at the appropriate place in the overall action sequence.

A series of examples will make ordering of actions clearer:

o If all actions have the same sequence number, regardless of whether it is '0' or non-zero, any order is acceptable.

o The values

```
1:ACTION A
2:ACTION B
1:ACTION C
3:ACTION D
```

indicate two acceptable orders: A,C,B,D or C,A,B,D, since A and C can be performed in either order, but only at the '1' position.

o The values

```
0:ACTION A
2:ACTION B
3:ACTION C
3:ACTION D
```

require that B,C, and D occur either as B,C,D or as B,D,C. Action A may appear at any point relative to B,C, and D. Thus the complete set of acceptable orders is: A,B,C,D; B,A,C,D; B,C,A,D; B,C,D,A; A,B,D,C; B,A,D,C; B,D,A,C; B,D,C,A.

Note that the non-zero sequence numbers need not start with '1', and they need not be consecutive. All that matters is their relative magnitude.

The property is defined as follows:

NAME	ActionOrder
DESCRIPTION	Unsigned integer indicating the relative position of an action in the sequence of actions aggregated by a policy rule.
SYNTAX	uint16

7.9. The Abstract Association "PolicyInSystem"

This abstract association inherits two object references from a higher- level CIM association class, Dependency. It overrides these object references to make them references to instances of the classes System and Policy. Subclasses of PolicyInSystem then override these object references again, to make them references to concrete policy classes.

The value of the abstract superclass is to convey that all subclasses have the same "dependency" semantics, and for ease of query to locate all policy "dependencies" on a System. These dependencies are related to scoping or hosting of the Policy.

The class definition for the association is as follows:

```
   NAME             PolicyInSystem
   DESCRIPTION      A generic association used to establish
                    dependency relationships between Policies and the
                    Systems that host them.
   DERIVED FROM     Dependency
   ABSTRACT         TRUE
   PROPERTIES       Antecedent[ref System[0..1]]
                    Dependent[ref Policy[0..n]]
```

7.10. The Weak Association "PolicyGroupInSystem"

This association links a PolicyGroup to the System in whose scope the PolicyGroup is defined.

The class definition for the association is as follows:

```
   NAME             PolicyGroupInSystem
   DESCRIPTION      A class representing the fact that a PolicyGroup
                    is defined within the scope of a System.
   DERIVED FROM     PolicyInSystem
   ABSTRACT         FALSE
   PROPERTIES       Antecedent[ref System[1..1]]
                    Dependent[ref PolicyGroup[weak]]
```

7.10.1. The Reference "Antecedent"

This property is inherited from PolicyInSystem, and overridden to restrict its cardinality to [1..1]. It serves as an object reference to a System that provides a scope for one or more PolicyGroups. Since this is a weak association, the cardinality for this object reference is always 1, that is, a PolicyGroup is always defined within the scope of exactly one System.

7.10.2. The Reference "Dependent"

This property is inherited from PolicyInSystem, and overridden to become an object reference to a PolicyGroup defined within the scope of a System. Note that for any single instance of the association class PolicyGroupInSystem, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given System may have 0, 1, or more than one PolicyGroups defined within its scope.

7.11. The Weak Association "PolicyRuleInSystem"

Regardless of whether it belongs to a PolicyGroup (or to multiple PolicyGroups), a PolicyRule is itself defined within the scope of a System. This association links a PolicyRule to the System in whose scope the PolicyRule is defined.

The class definition for the association is as follows:

```
  NAME             PolicyRuleInSystem
  DESCRIPTION      A class representing the fact that a PolicyRule
                   is defined within the scope of a System.
  DERIVED FROM     PolicyInSystem
  ABSTRACT         FALSE
  PROPERTIES       Antecedent[ref System[1..1]]
                   Dependent[ref PolicyRule[weak]]
```

7.11.1. The Reference "Antecedent"

This property is inherited from PolicyInSystem, and overridden to restrict its cardinality to [1..1]. It serves as an object reference to a System that provides a scope for one or more PolicyRules. Since this is a weak association, the cardinality for this object reference is always 1, that is, a PolicyRule is always defined within the scope of exactly one System.

7.11.2. The Reference "Dependent"

This property is inherited from PolicyInSystem, and overridden to become an object reference to a PolicyRule defined within the scope of a System. Note that for any single instance of the association class PolicyRuleInSystem, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given System may have 0, 1, or more than one PolicyRules defined within its scope.

7.12. The Association "PolicyConditionInPolicyRepository"

A reusable policy condition is always related to a single PolicyRepository, via the PolicyConditionInPolicyRepository association. This is not true for all PolicyConditions, however. An instance of PolicyCondition that represents a rule-specific condition is not related to any policy repository via this association.

The class definition for the association is as follows:

```
NAME            PolicyConditionInPolicyRepository
DESCRIPTION     A class representing the inclusion of a reusable
                PolicyCondition in a PolicyRepository.
DERIVED FROM    PolicyInSystem
ABSTRACT        FALSE
PROPERTIES      Antecedent[ref PolicyRepository[0..1]]
                Dependent[ref PolicyCondition[0..n]]
```

7.12.1. The Reference "Antecedent"

This property is inherited from PolicyInSystem, and overridden to become an object reference to a PolicyRepository containing one or more PolicyConditions. A reusable PolicyCondition is always related to exactly one PolicyRepository via the PolicyConditionInPolicyRepository association. The [0..1] cardinality for this property covers the two types of PolicyConditions: 0 for a rule-specific PolicyCondition, 1 for a reusable one.

7.12.2. The Reference "Dependent"

This property is inherited from PolicyInSystem, and overridden to become an object reference to a PolicyCondition included in a PolicyRepository. Note that for any single instance of the association class PolicyConditionInPolicyRepository, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given PolicyRepository may contain 0, 1, or more than one PolicyConditions.

7.13. The Association "PolicyActionInPolicyRepository"

A reusable policy action is always related to a single PolicyRepository, via the PolicyActionInPolicyRepository association. This is not true for all PolicyActions, however. An instance of PolicyAction that represents a rule-specific action is not related to any policy repository via this association.

The class definition for the association is as follows:

```
NAME            PolicyActionInPolicyRepository
DESCRIPTION     A class representing the inclusion of a reusable
                PolicyAction in a PolicyRepository.
DERIVED FROM    PolicyInSystem
ABSTRACT        FALSE
PROPERTIES      Antecedent[ref PolicyRepository[0..1]]
                Dependent[ref PolicyAction[0..n]]
```

7.13.1. The Reference "Antecedent"

This property is inherited from PolicyInSystem, and overridden to become an object reference to a PolicyRepository containing one or more PolicyActions. A reusable PolicyAction is always related to exactly one PolicyRepository via the PolicyActionInPolicyRepository association. The [0..1] cardinality for this property covers the two types of PolicyActions: 0 for a rule-specific PolicyAction, 1 for a reusable one.

7.13.2. The Reference "Dependent"

This property is inherited from PolicyInSystem, and overridden to become an object reference to a PolicyAction included in a PolicyRepository. Note that for any single instance of the association class PolicyActionInPolicyRepository, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given PolicyRepository may contain 0, 1, or more than one PolicyActions.

7.14. The Aggregation "PolicyRepositoryInPolicyRepository"

The PolicyRepositoryInPolicyRepository aggregation enables policy repositories to be nested. This derives from the higher level CIM association, CIM_SystemComponent, describing that Systems contain other ManagedSystemElements. This superclass could not be used for the other Policy aggregations, since Policies are not ManagedSystemElements, but ManagedElements. Note that it is assumed that this aggregation is used to form directed acyclic graphs and NOT ring structures.

The class definition for the aggregation is as follows:

```
NAME            PolicyRepositoryInPolicyRepository
DESCRIPTION     A class representing the aggregation of
                PolicyRepositories by a higher-level
                PolicyRepository.
DERIVED FROM    SystemComponent
ABSTRACT        FALSE
PROPERTIES      GroupComponent[ref PolicyRepository[0..n]]
                  PartComponent[ref PolicyRepository[0..n]]
```

7.14.1. The Reference "GroupComponent"

This property is inherited from the CIM class SystemComponent, and overridden to become an object reference to a PolicyRepository that contains one or more other PolicyRepositories. Note that for any single instance of the aggregation class PolicyRepositoryInPolicyRepository, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more than one PolicyRepositories that contain any given PolicyRepository.

7.14.2. The Reference "PartComponent"

This property is inherited from the CIM class SystemComponent, and overridden to become an object reference to a PolicyRepository contained by one or more other PolicyRepositories. Note that for any single instance of the aggregation class PolicyRepositoryInPolicyRepository, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given PolicyRepository may contain 0, 1, or more than one other PolicyRepositories.

8. Intellectual Property

9. Acknowledgements

The Policy Core Information Model in this document is closely based on the work of the DMTF's Service Level Agreements working group, so thanks are due to the members of that working group. Several of the policy classes in this model first appeared in early drafts on IPSec policy and QoS policy. The authors of these drafts were Partha Bhattacharya, Rob Adams, William Dixon, Roy Pereira, Raju Rajan, Jean-Christophe Martin, Sanjay Kamat, Michael See, Rajiv Chaudhury, Dinesh Verma, George Powers, and Raj Yavatkar. Some other elements of the model originated in work done by Yoram Snir, Yoram Ramberg, and Ron Cohen. In addition, we would like to thank Harald Alvestrand for conducting a thorough review of this document and providing many helpful suggestions, and Luis Sanchez and Russ Mundy for their help with the document's Security Considerations.

10. Security Considerations

The Policy Core Information Model (PCIM) presented in this document provides an object-oriented model for describing policy information. It provides a basic framework for describing the structure of policy information, in a form independent of any specific repository or access protocol, for use by an operational system. PCIM is not intended to represent any particular system design or implementation, nor does it define a protocol, and as such it does not have any specific security requirements.

However, it should also be noted that certain derivative documents, which use PCIM as a base, will need to convey more specific security considerations. In order to communicate the nature of what will be expected in these follow-on derivative documents, it is necessary to review the reasons that PCIM, as defined in this document, is neither implementable, nor representative of any real-world system, as well as the nature of the expected follow-on extensions and mappings.

There are three independent reasons that PCIM, as defined here, is neither implementable nor representative of any real-world system:

1. Its classes are independent of any specific repository that uses any specific access protocol. Therefore, its classes are designed not to be implemented directly. PCIM should instead be viewed as a schematic that directs how information should be represented, independent of any specific model implementation constraints.

2. Its classes were designed to be independent of any specific policy domain. For example, DiffServ and IPSec represent two different policy domains. Each document which extends PCIM to one of these domains will derive subclasses from the classes and relationships defined in PCIM, in order to represent extensions of a generic model to cover specific technical domains.

3. It's an information model, which must be mapped to a specific data model (native CIM schema, LDAP schema, MIB, whatever) before it can be implemented. Derivative documents will map the extended information models noted in item 2, above, to specific types of data model implementations.

Even though specific security requirements are not appropriate for PCIM, specific security requirements MUST be defined for each operational real- world application of PCIM. Just as there will be a wide range of operational, real-world systems using PCIM, there will also be a wide range of security requirements for these systems. Some operational, real-world systems that are deployed using PCIM may have extensive security requirements that impact nearly all classes and subclasses utilized by such a system, while other systems' security requirements might have very little impact.

The derivative documents, discussed above, will create the context for applying operational, real-world, system-level security requirements against the various models which derive from PCIM.

For example, in some real-world scenarios, the values associated with certain properties, within certain instantiated classes, may represent information associated with scarce, and/or costly (and therefore valuable) resources. It may be the case that these values must not be disclosed to, or manipulated by, unauthorized parties. As long as the derived model remains an information model (as opposed to a data model), it is not possible to discuss the data model-specific tools and mechanisms that are available for achieving the authentication and authorization implicit in a requirement that restricts read and/or read- write access to these values. Therefore, these mechanisms will need to be discussed in each of the data models to which the derived information models are mapped. If there are any general security requirements that can be identified and can be applied across multiple types of data models, it would be appropriate to discuss those at the information model level, rather than the data model level. In any case, any identified security requirements that are not dealt with in the information model document, MUST be dealt with in the derivative data model documents.

We can illustrate these points by extending the example from Section 2. A real-world system that provides QoS Gold Service to John would likely need to provide at least the following security-related capabilities and mechanisms (see [12] for definitions of security related terms):

o Data integrity for the information (e.g., property values and instantiated relationships) that specify that John gets QoS Gold Service, from the point(s) that the information is entered into the system to the point(s) where network components actually provide that Service.

o Authentication and Authorization methods to ensure that only system administrators (and not John or other engineers) can remotely administer components of the system.

o An Authentication method to insure that John receives Gold Service, and the other members of the engineering group receive Bronze Service.

These are one possible set of requirements associated with an example real-world system which delivers Gold Service, and the appropriate place to document these would be in some combination of the information model and the derivative data models for QoS Policy. Each of the data models would also need to discuss how these requirements are satisfied, using the mechanisms typically available to such a data model, given the particular technology or set of technologies which it may employ.

11. References

[1] Distributed Management Task Force, Inc., "DMTF Technologies: CIM Standards << CIM Schema: Version 2.4", available via links on the following DMTF web page: http://www.dmtf.org/spec/cim_schema_v24.html.

[2] Distributed Management Task Force, Inc., "Common Information Model (CIM) Specification, version 2.2, June 1999. This document is available on the following DMTF web page: http://www.dmtf.org/spec/cims.html.

[3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[4] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[5] J. Strassner and S. Judd, "Directory-Enabled Networks", version 3.0c5 (August 1998). A PDF file is available at http://www.murchiso.com/den/#denspec.

[6] J. Strassner, policy architecture BOF presentation, 42nd IETF Meeting, Chicago, Illinois, October, 1998. Minutes of this BOF are available at the following location: http://www.ietf.org/proceedings/98aug/index.html.

[7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[8] Levi, D. and J. Schoenwaelder, "Definitions of Managed Objects for Scheduling Management Operations", RFC 2591, May 1999.

[9] Yavatkar, R., Pendarakis, D. and R. Guerin, "A Framework for Policy-based Admission Control", RFC 2753, January 2000.

[10] Dawson, F. and D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 2445, November 1998.

[11] Strassner, J., and E. Ellesson, B. Moore, R. Moats, "Policy Core LDAP Schema", Work in Progress.

[12] Shirey, R., "Internet Security Glossary", FYI 36, RFC 2828, May 2000.

Note: the CIM 2.4 Schema specification is defined by the following set of MOF files, available from the following URL:

http://www.dmtf.org/spec/CIM_Schema24/CIM_Schema24.zip

12. Authors' Addresses

Ed Ellesson
LongBoard, Inc.
2505 Meridian Pkwy, #100
Durham, NC 27713

Phone: +1 919-361-3230
Fax: +1 919-361-3299
EMail: eellesson@lboard.com

Bob Moore
IBM Corporation, BRQA/502
4205 S. Miami Blvd.
Research Triangle Park, NC 27709

Phone: +1 919-254-4436
Fax: +1 919-254-6243
EMail: remoore@us.ibm.com

John Strassner
Cisco Systems, Bldg 15
170 West Tasman Drive
San Jose, CA 95134

Phone: +1 408-527-1069
Fax: +1 408-527-6351
EMail: johns@cisco.com

Andrea Westerinen
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134

Phone: +1 408-853-8294
Fax: +1 408-527-6351
EMail: andreaw@cisco.com

13. Appendix A: Class Identification in a Native CIM Implementation

While the CommonName property is present in the abstract superclass Policy, and is thus available in all of its instantiable subclasses, CIM does not use this property for naming instances. The following subsections discuss how naming is handled in a native CIM implementation for each of the instantiable classes in the Policy Core Information Model.

Two things should be noted regarding CIM naming:

- When a CIM association is specified as "weak", this is a statement about naming scopes: an instance of the class at the weak end of the association is named within the scope of an instance of the class at the other end of the association. This is accomplished by propagation of keys from the instance of the scoping class to the instance of the weak class. Thus the weak class has, via key propagation, all the keys from the scoping class, and it also has one or more additional keys for distinguishing instances of the weak class, within the context of the scoping class.

- All class names in CIM are limited to alphabetic and numeric characters plus the underscore, with the restriction that the first character cannot be numeric. Refer to Appendix F "Unicode Usage" in reference [2] for an exact specification of how CIM class names are encoded in CIM strings.

13.1. Naming Instances of PolicyGroup and PolicyRule

A policy group always exists in the context of a system. In the Policy Core Information Model, this is captured by the weak aggregation PolicyGroupInSystem between a PolicyGroup and a System. Note that System serves as the base class for describing network devices and administrative domains.

A policy rule also exists in the context of a system. In the Policy Core Information Model, this is captured by the weak association PolicyRuleInSystem between a PolicyRule and a System.

The following sections define the CIM keys for PolicyGroup and PolicyRule.

13.1.1. PolicyGroup's CIM Keys

The CIM keys of the PolicyGroup class are:

- SystemCreationClassName (A CIM_System key, propagated due to the weak association, PolicyGroupInSystem)

```
   o  SystemName (A CIM_System key, propagated due to  the weak
      association, PolicyGroupInSystem)
   o  CreationClassName
   o  PolicyGroupName

   They are defined in Reference [1] as follows:

   NAME           SystemCreationClassName
   DESCRIPTION    SystemCreationClassName represents the class name of
                  the CIM System object providing the naming scope for
                  the instance of PolicyGroup.
   SYNTAX         string [MaxLen 256]
   QUALIFIER      key

   NAME           SystemName
   DESCRIPTION    SystemName represent the individual name of the
                  particular System object, providing the naming scope
                  for the instance of PolicyGroup.
   SYNTAX         string [MaxLen 256]
   QUALIFIER      key

   NAME           CreationClassName
   DESCRIPTION    This property is set to "CIM_PolicyGroup", if the
                  PolicyGroup object is directly instantiated.  Or, it
                  is equal to the class name of the PolicyGroup
                  subclass that is instantiated.
   SYNTAX         string [MaxLen 256]
   QUALIFIER      key

   NAME           PolicyGroupName
   DESCRIPTION    The identifying name of this policy group.
   SYNTAX         string [MaxLen 256]
   QUALIFIER      key
```

13.1.2. PolicyRule's CIM Keys

The CIM keys of the PolicyRule class are:

- o SystemCreationClassName (A CIM_System key, propagated due to the weak association PolicyRuleInSystem)
- o SystemName (A CIM_System key, propagated due to the weak association PolicyRuleInSystem)
- o CreationClassName
- o PolicyRuleName

SystemCreationClassName and SystemName work the same as defined for the class PolicyGroup. See Section 13.1.1 for details.

The other two properties are defined in Reference [1] as follows:

NAME	CreationClassName
DESCRIPTION	This property is set to "CIM_PolicyRule", if the PolicyRule object is directly instantiated. Or, it is equal to the class name of the PolicyRule subclass that is instantiated.
SYNTAX	string [MaxLen 256]
QUALIFIER	key

NAME	PolicyRuleName
DESCRIPTION	The identifying name of this policy rule.
SYNTAX	string [MaxLen 256]
QUALIFIER	key

13.2. Naming Instances of PolicyCondition and Its Subclasses

The CIM keys of the PolicyCondition class are:

- o SystemCreationClassName
- o SystemName
- o PolicyRuleCreationClassName
- o PolicyRuleName
- o CreationClassName
- o PolicyConditionName

Note that none of the keys are defined as propagated, although they appear to fit this convention. The reason for this difference is because (as indicated in Sections 5.1 and 6.4) the PolicyCondition class is used to represent both reusable and rule-specific conditions. This, in turn, affects what associations are valid for an instance of PolicyCondition, and how that instance is named.

In an ideal world, an instance of the PolicyCondition class would be scoped either by its PolicyRepository (for a reusable condition) or by its PolicyRule (for a rule-specific condition). However, CIM has the restriction that a given class can only be "weak" to one other class (i.e., defined by one weak association).

To work within the restrictions of CIM naming, it is necessary to "simulate" weak associations between PolicyCondition and PolicyRule, and between PolicyCondition and PolicyRepository, through a technique we'll call manual key propagation. Strictly speaking, manual key propagation isn't key propagation at all. But it has the same effect as (true) key propagation, so the name fits.

Figure 9 illustrates how manual propagation works in the case of PolicyCondition. (Note that only the key properties are shown for each of the classes.) In the figure, the line composed of 'I's indicates class inheritance, the one composed of 'P's indicates (true) key propagation via the weak aggregation PolicyRuleInSystem, and the ones composed of 'M's indicate manual key propagation.

```
   +-------------------+
   |      System       |
   +-------------------+
   |CreationClassName  |
   |Name               |
   +-------------------+
             ^     P
             I     PPPPPPPPPPPPPPPPPPPPPPPPPPPPP
             I                                 P
   +-------------------+      +----------------v--------------+
   |    AdminDomain    |      |         PolicyRule            |
   +-------------------+      +-------------------------------+
   |CreationClassName  |      | System.CreationClassName      |
   |Name               |      | System.Name                   |
   +-------------------+      | CreationClassName             |
             ^                | PolicyRuleName                |
             I                +-------------------------------+
             I                         M
             I                         M
   +-------------------+               M
   | PolicyRepository  |               M
   +-------------------+               M
   |CreationClassName  |               M
   |Name               |               M
   +-------------------+               M
                   M                   M
                   M                   M
                   M                   M
              +----v-------------------v-----+
              |       PolicyCondition        |
              +------------------------------+
              | SystemCreationClassName      |
              | SystemName                   |
              | PolicyRuleCreationClassName  |
              | PolicyRuleName               |
              | CreationClassName            |
              | PolicyConditionName          |
              +------------------------------+
```

Figure 9. Manual Key Propagation for Naming PolicyConditions

Looking at Figure 9, we see that two key properties, CreationClassName and Name, are defined in the System class, and inherited by its subclasses AdminDomain and PolicyRepository. Since PolicyRule is weak to System, these two keys are propagated to it; it also has its own keys CreationClassName and PolicyRuleName.

A similar approach, though not automatic, is used in "manual key propagation". Here is the approach for rule-specific and reusable PolicyConditions:

o The manual propagation of keys from PolicyRule to PolicyCondition involves copying the values of PolicyRule's four key properties into four similarly named key properties in PolicyCondition. From the point of view of the CIM specification language, the property SystemName in PolicyCondition is a completely new key property. However, the relationship to the Name property in System is defined in the description of SystemName.

o The manual propagation of keys from PolicyRepository to PolicyCondition works in exactly the same way for the first two key properties. However, since PolicyRepository doesn't include PolicyRule properties, the PolicyRuleCreationClassName and PolicyRuleName have no values. A special value, "No Rule", is assigned to both of these properties in this case, indicating that this instance of PolicyCondition is not named within the scope of any particular policy rule.

The following section defines the specific CIM keys for PolicyCondition.

13.2.1. PolicyCondition's CIM Keys

PolicyCondition's key properties are defined in Reference [1] as follows:

NAME	SystemCreationClassName
DESCRIPTION	SystemCreationClassName represents the class name of the CIM System object providing the naming scope for the instance of PolicyCondition. For a rule-specific policy condition, this is the type of system (e.g., the name of the class that created this instance) in whose context the policy rule is defined. For a reusable policy condition, this is set to "CIM_PolicyRepository", if the PolicyRepository object is directly instantiated. Or, it is equal to the class name of the PolicyRepository subclass that is instantiated.
SYNTAX	string [MaxLen 256]

```
   QUALIFIER       key

   NAME            SystemName
   DESCRIPTION     The name of the System object in whose scope this
                   policy condition is defined.  This property
                   completes the identification of the System object.
                   For a rule-specific policy condition, this is the
                   name of the instance of the system in whose
                   context the policy rule is defined.  For a
                   reusable policy condition, this is name of the
                   instance of PolicyRepository that holds the policy
                   condition.
   SYNTAX          string [MaxLen 256]
   QUALIFIER       key

   NAME            PolicyRuleCreationClassName
   DESCRIPTION     For a rule-specific policy condition, this
                   property identifies the class name of the policy
                   rule instance, in whose scope this instance of
                   PolicyCondition exists.  For a reusable policy
                   condition, this property is set to a special
                   value, "No Rule", indicating that this instance
                   of PolicyCondition is not unique to one policy
                   rule.
   SYNTAX          string [MaxLen 256]
   QUALIFIER       key

   NAME            PolicyRuleName
   DESCRIPTION     For a rule-specific policy condition,
                   PolicyRuleName completes the identification of
                   the PolicyRule object with which this condition
                   is associated.  For a reusable policy condition,
                   a special value, "No Rule", is used to indicate
                   that this condition is reusable.
   SYNTAX          string [MaxLen 256]
   QUALIFIER       key

   NAME            CreationClassName
   DESCRIPTION     The class name of the PolicyCondition subclass
                   that is instantiated.
   SYNTAX          string [MaxLen 256]
   QUALIFIER       key

   NAME            PolicyConditionName
   DESCRIPTION     The identifying name of this policy condition.
   SYNTAX          string [MaxLen 256]
   QUALIFIER       key
```

13.3. Naming Instances of PolicyAction and Its Subclasses

From the point of view of naming, the PolicyAction class and its subclasses work exactly like the PolicyCondition class and its subclasses. See Section 13.2 and 13.2.1 for details.

Specifically, the CIM keys of PolicyAction are:

- o SystemCreationClassName
- o SystemName
- o PolicyRuleCreationClassName
- o PolicyRuleName
- o CreationClassName
- o PolicyActionName

They are defined in Reference [1] as follows:

```
NAME              SystemCreationClassName
DESCRIPTION       SystemCreationClassName represents the class name
                  of the CIM System object providing the naming
                  scope for the instance of PolicyAction.  For a
                  rule-specific policy action, this is the type of
                  system (e.g., the name of the class that created
                  this instance) in whose context the policy rule
                  is defined.  For a reusable policy action, this
                  is set to "CIM_PolicyRepository", if the
                  PolicyRepository object is directly instantiated.
                  Or, it is equal to the class name of the
                  PolicyRepository subclass that is instantiated.
SYNTAX            string [MaxLen 256]
QUALIFIER         key

NAME              SystemName
DESCRIPTION       The name of the System object in whose scope this
                  policy action is defined.  This property completes
                  the identification of the System object.  For a
                  rule-specific policy action, this is the name of
                  the instance of the system in whose context the
                  policy rule is defined.  For a reusable policy
                  action, this is name of the instance of
                  PolicyRepository that holds the policy action.
SYNTAX            string [MaxLen 256]
QUALIFIER         key

NAME              PolicyRuleCreationClassName
DESCRIPTION       For a rule-specific policy action, this property
                  identifies the class name of the policy rule
                  instance, in whose scope this instance of
```

```
                     PolicyAction exists.  For a reusable policy
                     action, this property is set to a special value,
                     "No Rule", indicating that this instance of
                     PolicyAction is not unique to one policy rule.
   SYNTAX            string [MaxLen 256]
   QUALIFIER         key

   NAME              PolicyRuleName
   DESCRIPTION       For a rule-specific policy action, PolicyRuleName
                     completes the identification of the PolicyRule
                     object with which this action is associated.  For
                     a reusable policy action, a special value, "No
                     Rule", is used to indicate that this action is
                     reusable.
   SYNTAX            string [MaxLen 256]
   QUALIFIER         key

   NAME              CreationClassName
   DESCRIPTION       The class name of the PolicyAction subclass that is
                     instantiated.
   SYNTAX            string [MaxLen 256]
   QUALIFIER         key

   NAME              PolicyActionName
   DESCRIPTION       The identifying name of this policy action.
   SYNTAX            string [MaxLen 256]
   QUALIFIER         key
```

13.4. Naming Instances of PolicyRepository

An instance of PolicyRepository is named by the two key properties CreationClassName and Name that it inherits from its superclass AdminDomain. These properties are actually defined in AdminDomain's superclass, System, and then inherited by AdminDomain.

For instances of PolicyRepository itself, the value of CreationClassName must be "CIM_PolicyRepository". (Recall that for readability the prefix "CIM_" has been omitted from all class names in this document). If a subclass of PolicyRepository (perhaps QosPolicyRepository) is defined and instantiated, then the class name "CIM_QosPolicyRepository" is used in CreationClassName.

The Name property simply completes the identification of the instance of PolicyRepository.

13.5. Role of the CreationClassName Property in Naming

To provide for more flexibility in instance naming, CIM makes use of a property called CreationClassName. The idea of CreationClassName is to provide another dimension that can be used to avoid naming collisions, in the specific case of instances belonging to two different subclasses of a common superclass. An example will illustrate how CreationClassName works.

Suppose we have instances of two different subclasses of PolicyCondition, FrameRelayPolicyCondition and BgpPolicyCondition, and that these instances apply to the same context. If we had only the single key property PolicyConditionName available for distinguishing the two instances, then a collision would result from naming both of the instances with the key value PCName = "PC-1". Thus policy administrators from widely different disciplines would have to coordinate their naming of PolicyConditions for this context.

With CreationClassName, collisions of this type can be eliminated, without requiring coordination among the policy administrators. The two instances can be distinguished by giving their CreationClassNames different values. One instance is now identified with the two keys

CreationClassName = "FrameRelayPolicyCondition" + PCName = "PC-1",

while the other is identified with

CreationClassName = "BgpPolicyCondition" + PCName = "PC-1".

Each of the instantiable classes in the Core Model includes the CreationClassName property as a key in addition to its own class-specific key property.

13.6. Object References

Today, all CIM associations involve two object references. CIM decomposes an object reference into two parts: a high-order part that identifies an object manager and namespace, and a model path that identifies an object instance within a namespace. The model path, in turn, can be decomposed into an object class identifier and a set of key values needed to identify an instance of that class.

Because the object class identifier is part of the model path, a CIM object reference is strongly typed. The GroupComponent object reference in the PolicyGroupInPolicyGroup association, for example, can only point to an instance of PolicyGroup, or to an instance of a subclass of PolicyGroup. Contrast this with LDAP, where a DN pointer is completely untyped: it identifies (by DN) an entry, but places no restriction on that entry's object class(es).

An important difference between CIM property definitions and LDAP attribute type definitions was identified earlier in Section 6: while an LDAP attribute type definition has global scope, a CIM property definition applies only to the class in which it is defined. Thus properties having the same name in two different classes are free to have different data types. CIM takes advantage of this flexibility by allowing the data type of an object reference to be overridden in a subclass of the association class in which it was initially defined.

For example, the object reference GroupComponent is defined in the abstract aggregation class PolicyComponent to be a reference to an instance of the class Policy. This data type for GroupComponent is then overridden in subclasses of PolicyComponent. In PolicyGroupInPolicyGroup, for example, GroupComponent becomes a reference to an instance of PolicyGroup. But in PolicyConditionInPolicyRule it becomes a reference to an instance of PolicyRule. Of course there is not total freedom in this overriding of object references. In order to remain consistent with its abstract superclass, a subclass of PolicyComponent can only override GroupComponent to be a reference to a subclass of Policy. A Policy class is the generic context for the GroupComponent reference in PolicyComponent.

14. Appendix B: The Core Policy MOF

```
// ==================================================================
// Title:     Core Policy MOF Specification 2.4
// Filename:  CIM_Policy24.MOF
// Version:   2.4
// Release:   0
// Description: The object classes below are listed in an order that
//              avoids forward references.  Required objects, defined
//       by other working groups, are omitted.
// Date: 06/27/2000
//     CIMCR516a - Rooted the model associations under Policy
//       Component or PolicyInSystem.  Corrected PolicyCondition/
//       PolicyActionInPolicyRepository to subclass from
//       PolicyInSystem (similar to Groups and Roles 'InSystem')
// ==================================================================
// Author:    DMTF SLA (Service Level Agreement) Working Group
// ==================================================================
// Pragmas
// ==================================================================
#pragma Locale ("en-US")


// ==================================================================
// Policy
// ==================================================================
   [Abstract, Description (
         "An abstract class describing common properties of all "
         "policy rule-related subclasses, such as PolicyGroup, Policy"
         "Rule and PolicyCondition. All instances of policy rule-"
         "related entities will be created from subclasses of CIM_"
         "Policy.  The exception to this statement is PolicyRepository "
         "which is a type of CIM_System.")
   ]
class CIM_Policy : CIM_ManagedElement
{
      [Description (
         "A user-friendly name of this policy-related object.")
      ]
   string CommonName;
      [Description (
         "An array of keywords for characterizing / categorizing "
         "policy objects.  Keywords are of one of two types: \n"
         "  o Keywords defined in this and other MOFs, or in DMTF "
         "    white papers.  These keywords provide a vendor-"
         "    independent, installation-independent way of "
         "    characterizing policy objects. \n"
         "  o Installation-dependent keywords for characterizing "
```

```
        "   policy objects.  Examples include 'Engineering', "
        "   'Billing', and 'Review in December 2000'.  \n"
        "This MOF defines the following keywords:  'UNKNOWN', "
        "'CONFIGURATION', 'USAGE', 'SECURITY', 'SERVICE', "
        "'MOTIVATIONAL', 'INSTALLATION', and 'EVENT'.  These "
        "concepts are self-explanatory and are further discussed "
        "in the SLA/Policy White Paper.  One additional keyword "
        "is defined: 'POLICY'.  The role of this keyword is to "
        "identify policy-related instances that may not be otherwise "
        "identifiable, in some implementations.  The keyword 'POLICY' "
        "is NOT mutually exclusive of the other keywords "
        "specified above.")
     ]
   string PolicyKeywords [];
};

// ==================================================================
//    PolicyComponent
// ==================================================================
   [Association, Abstract, Aggregation, Description (
        "CIM_PolicyComponent is a generic association used to "
        "establish 'part of' relationships between the subclasses of "
        "CIM_Policy.  For example, the PolicyConditionInPolicyRule "
        "association defines that PolicyConditions are part of a "
        "PolicyRule.")
   ]
class CIM_PolicyComponent
{
       [Aggregate, Key, Description (
         "The parent Policy in the association.")
       ]
    CIM_Policy REF GroupComponent;
       [Key, Description (
         "The child/part Policy in the association.")
       ]
    CIM_Policy REF PartComponent;
};

// ==================================================================
//    PolicyInSystem
// ==================================================================
   [Association, Abstract, Description (
        "  CIM_PolicyInSystem is a generic association used to "
        "establish dependency relationships between Policies and the "
        "Systems that host them.  These Systems may be ComputerSystems "
        "where Policies are 'running' or they may be Policy"
        "Repositories where Policies are stored.  This relationship "
        "is similar to the concept of CIM_Services being dependent "
```

```
          "on CIM_Systems as defined by the HostedService "
          "association.  \n"
          "  Cardinality is Max(1) for the Antecedent/System "
          "reference since Policies can only be hosted in at most one "
          "System context.  Some subclasses of the association will "
          "further refine this definition to make the Policies Weak "
          "to Systems.  Other subclasses of PolicyInSystem will "
          "define an optional hosting relationship.  Examples of each "
          "of these are the PolicyRuleInSystem and PolicyConditionIn"
          "PolicyRepository associations, respectively.")
   ]
class CIM_PolicyInSystem : CIM_Dependency
{
       [Override ("Antecedent"), Max (1), Description (
          "The hosting System.")
       ]
    CIM_System REF Antecedent;
       [Override ("Dependent"), Description (
          "The hosted Policy.")
       ]
    CIM_Policy REF Dependent;
};

// ==================================================================
// PolicyGroup
// ==================================================================
   [Description (
         "A container for either a set of related PolicyGroups "
         "or a set of related PolicyRules, but not both.  Policy"
         "Groups are defined and named relative to the CIM_System "
         "which provides their context.")
   ]
class CIM_PolicyGroup : CIM_Policy
{
      [Propagated("CIM_System.CreationClassName"),
         Key, MaxLen (256),
         Description ("The scoping System's CreationClassName.")
      ]
   string SystemCreationClassName;
      [Propagated("CIM_System.Name"),
         Key, MaxLen (256),
         Description ("The scoping System's Name.")
      ]
   string SystemName;
      [Key, MaxLen (256), Description (
         "CreationClassName indicates the name of the class or the "
         "subclass used in the creation of an instance.  When used "
         "with the other key properties of this class, this property "
         "allows all instances of this class and its subclasses to "
         "be uniquely identified.") ]
   string CreationClassName;
      [Key, MaxLen (256), Description (
         "A user-friendly name of this PolicyGroup.")
      ]
   string PolicyGroupName;
};

// ==================================================================
//    PolicyGroupInPolicyGroup
// ==================================================================
   [Association, Aggregation, Description (
         "A relationship that aggregates one or more lower-level "
         "PolicyGroups into a higher-level Group.  A Policy"
         "Group may aggregate either PolicyRules or other Policy"
         "Groups, but not both.")
   ]
class CIM_PolicyGroupInPolicyGroup : CIM_PolicyComponent
{
        [Override ("GroupComponent"), Aggregate, Description (
         "A PolicyGroup that aggregates other Groups.")
        ]
    CIM_PolicyGroup REF GroupComponent;
        [Override ("PartComponent"), Description (
         "A PolicyGroup aggregated by another Group.")
        ]
    CIM_PolicyGroup REF PartComponent;
};

// ==================================================================
//    PolicyGroupInSystem
// ==================================================================
   [Association, Description (
         "An association that links a PolicyGroup to the System "
         "in whose scope the Group is defined.")
   ]
class CIM_PolicyGroupInSystem : CIM_PolicyInSystem
{
        [Override ("Antecedent"), Min(1), Max(1), Description (
         "The System in whose scope a PolicyGroup is defined.")
        ]
    CIM_System REF Antecedent;
        [Override ("Dependent"), Weak, Description (
         "A PolicyGroup named within the scope of a System.")
        ]
    CIM_PolicyGroup REF Dependent;
};
```

```
// ==================================================================
// PolicyRule
// ==================================================================
   [Description (
        "  The central class for representing the 'If Condition then "
         "Action' semantics associated with a policy rule. "
         "A PolicyRule condition, in the most general sense, is "
         "represented as either an ORed set of ANDed conditions "
         "(Disjunctive Normal Form, or DNF) or an ANDed set of ORed "
         "conditions (Conjunctive Normal Form, or CNF). Individual "
         "conditions may either be negated (NOT C) or unnegated (C). "
         "The actions specified by a PolicyRule are to be performed "
         "if and only if the PolicyRule condition (whether it is "
         "represented in DNF or CNF) evaluates to TRUE.\n\n"
         "  "
         "The conditions and actions associated with a PolicyRule "
         "are modeled, respectively, with subclasses of Policy"
         "Condition and PolicyAction.  These condition and action "
         "objects are tied to instances of PolicyRule by the Policy"
         "ConditionInPolicyRule and PolicyActionInPolicyRule "
         "aggregations.\n\n"
         "  "
         "A PolicyRule may also be associated with one or more policy "
         "time periods, indicating the schedule according to which the "
         "policy rule is active and inactive.  In this case it is the "
         "PolicyRuleValidityPeriod aggregation that provides this "
         "linkage.\n\n"
         "  "
         "The PolicyRule class uses the property ConditionListType, to "
         "indicate whether the conditions for the rule are in DNF or "
         "CNF.  The PolicyConditionInPolicyRule aggregation contains "
         "two additional properties to complete the representation of "
         "the Rule's conditional expression.  The first of these "
         "properties is an integer to partition the referenced "
         "PolicyConditions into one or more groups, and the second is a "
         "Boolean to indicate whether a referenced Condition is "
         "negated.  An example shows how ConditionListType and these "
         "two additional properties provide a unique representation "
         "of a set of PolicyConditions in either DNF or CNF.\n\n"
         "  "
         "Suppose we have a PolicyRule that aggregates five "
         "PolicyConditions C1  through C5, with the following values "
         "in the properties of the five PolicyConditionInPolicyRule "
         "associations:\n"
        "    C1:  GroupNumber = 1, ConditionNegated = FALSE\n "
        "    C2:  GroupNumber = 1, ConditionNegated = TRUE\n  "
        "    C3:  GroupNumber = 1, ConditionNegated = FALSE\n "
        "    C4:  GroupNumber = 2, ConditionNegated = FALSE\n "
```

```
         "    C5:  GroupNumber = 2, ConditionNegated = FALSE\n\n "
         "  "
         "If ConditionListType = DNF, then the overall condition for "
         "the PolicyRule is:\n"
         "         (C1 AND (NOT C2) AND C3) OR (C4 AND C5)\n\n"
         "  "
         "On the other hand, if ConditionListType = CNF, then the "
         "overall condition for the PolicyRule is:\n"
         "         (C1 OR (NOT C2) OR C3) AND (C4 OR C5)\n\n"
         "  "
         "In both cases, there is an unambiguous specification of "
         "the overall condition that is tested to determine whether "
         "to perform the PolicyActions associated with the PolicyRule.")
   ]
class CIM_PolicyRule : CIM_Policy
{
        [Propagated("CIM_System.CreationClassName"),
         Key, MaxLen (256),
         Description ("The scoping System's CreationClassName.")
        ]
    string SystemCreationClassName;
        [Propagated("CIM_System.Name"),
         Key, MaxLen (256),
         Description ("The scoping System's Name.")
        ]
    string SystemName;
        [Key, MaxLen (256), Description (
           "CreationClassName indicates the name of the class or the "
           "subclass used in the creation of an instance.  When used "
           "with the other key properties of this class, this property "
           "allows all instances of this class and its subclasses to "
           "be uniquely identified.") ]
    string CreationClassName;
        [Key, MaxLen (256), Description (
           "A user-friendly name of this PolicyRule.")
        ]
    string PolicyRuleName;
        [Description (
           "Indicates whether this PolicyRule is administratively "
           "enabled, administratively disabled, or enabled for "
           "debug.  When the property has the value 3 (\"enabledFor"
           "Debug\"), the entity evaluating the PolicyConditions is "
           "instructed to evaluate the conditions for the Rule, but not "
           "to perform the actions if the PolicyConditions evaluate to "
           "TRUE.  This serves as a debug vehicle when attempting to "
           "determine what policies would execute in a particular "
           "scenario, without taking any actions to change state "
           "during the debugging.  The default value is 1
(\"enabled\")."),
         ValueMap { "1", "2", "3" },
         Values { "enabled", "disabled", "enabledForDebug" }
        ]
    uint16 Enabled;
        [Description (
           "Indicates whether the list of PolicyConditions "
           "associated with this PolicyRule is in disjunctive "
           "normal form (DNF) or conjunctive normal form (CNF)."
           "The default value is 1 (\"DNF\")."),
         ValueMap { "1", "2" },
         Values { "DNF", "CNF" }
        ]
    uint16 ConditionListType;
        [Description (
           "A free-form string that can be used to provide "
           "guidelines on how this PolicyRule should be used.")
        ]
    string RuleUsage;
        [Description (
           "A non-negative integer for prioritizing this Policy"
           "Rule relative to other Rules.  A larger value "
           "indicates a higher priority.  The default value is 0.")
        ]
    uint16 Priority;
        [Description (
           "A flag indicating that the evaluation of the Policy"
           "Conditions and execution of PolicyActions (if the "
           "Conditions evaluate to TRUE) is required.  The "
           "evaluation of a PolicyRule MUST be attempted if the "
           "Mandatory property value is TRUE.  If the Mandatory "
           "property is FALSE, then the evaluation of the Rule "
           "is 'best effort' and MAY be ignored.")
        ]
    boolean Mandatory;
        [Description (
           "This property gives a policy administrator a way "
           "of specifying how the ordering of the PolicyActions "
           "associated with this PolicyRule is to be interpreted. "
           "Three values are supported:\n"
           "  o mandatory(1): Do the actions in the indicated "
           "    order, or don't do them at all.\n"
           "  o recommended(2): Do the actions in the indicated "
           "    order if you can, but if you can't do them in this "
           "    order, do them in another order if you can.\n"
           "  o dontCare(3): Do them -- I don't care about the "
           "    order.\n"
           "The default value is 3 (\"dontCare\")."),
          ValueMap { "1", "2", "3" },
          Values { "mandatory", "recommended", "dontCare" }
         ]
     uint16 SequencedActions;
         [Description (
          "This property represents the roles and role combinations "
          "associated with a PolicyRule.  Each value represents one "
          "role or role combination.  Since this is a multi-valued "
          "property, more than one role or combination can be associated "
          "with a single policy rule.  Each value is a string of the "
          "form:\n"
          "  <RoleName>[&&<RoleName>]*\n"
          "where the individual role names appear in alphabetical order "
          "(according to the collating sequence for UCS-2).")
         ]
     string PolicyRoles [];
};

// ==================================================================
//    PolicyRuleInPolicyGroup
// ==================================================================
   [Association, Aggregation, Description (
         "A relationship that aggregates one or more PolicyRules "
         "into a PolicyGroup.  A PolicyGroup may aggregate either "
         "PolicyRules or other PolicyGroups, but not both.")
   ]
class CIM_PolicyRuleInPolicyGroup : CIM_PolicyComponent
{
        [Override ("GroupComponent"), Aggregate, Description (
         "A PolicyGroup that aggregates one or more PolicyRules.")
        ]
    CIM_PolicyGroup REF GroupComponent;
        [Override ("PartComponent"), Description (
         "A PolicyRule aggregated by a PolicyGroup.")
        ]
    CIM_PolicyRule REF PartComponent;
};

// ==================================================================
//    PolicyRuleInSystem
// ==================================================================
   [Association, Description (
         "An association that links a PolicyRule to the System "
         "in whose scope the Rule is defined.")
   ]
class CIM_PolicyRuleInSystem : CIM_PolicyInSystem
{
        [Override ("Antecedent"), Min(1), Max(1), Description (
          "The System in whose scope a PolicyRule is defined.")
         ]
     CIM_System REF Antecedent;
         [Override ("Dependent"), Weak, Description (
          "A PolicyRule named within the scope of a System.")
         ]
     CIM_PolicyRule REF Dependent;
};

// ==================================================================
// PolicyRepository
// ==================================================================
   [Description (
         "A class representing an administratively defined "
         "container for reusable policy-related information. "
         "This class does not introduce any additional "
         "properties beyond those in its superclass "
         "AdminDomain.  It does, however, participate in a "
         "number of unique associations."
         "\n\n"
         "An instance of this class uses the NameFormat value"
         "\"PolicyRepository\", which is defined in the AdminDomain"
         "class.")
   ]
class CIM_PolicyRepository : CIM_AdminDomain
{
};

// ==================================================================
//    PolicyRepositoryInPolicyRepository
// ==================================================================
   [Association, Aggregation, Description (
         "A relationship that aggregates one or more lower-level "
         "PolicyRepositories into a higher-level Repository.")
   ]
class CIM_PolicyRepositoryInPolicyRepository : CIM_SystemComponent
{
        [Override ("GroupComponent"), Aggregate, Description (
         "A PolicyRepository that aggregates other Repositories.")
        ]
    CIM_PolicyRepository REF GroupComponent;
        [Override ("PartComponent"), Description (
         "A PolicyRepository aggregated by another Repository.")
        ]
    CIM_PolicyRepository REF PartComponent;
};

// ==================================================================
// PolicyCondition
// ==================================================================
   [Abstract, Description (
         "A class representing a rule-specific or reusable policy "
         "condition to be evaluated in conjunction with a Policy"
         "Rule.  Since all operational details of a PolicyCondition "
         "are provided in subclasses of this object, this class is "
         "abstract.")
   ]
class CIM_PolicyCondition : CIM_Policy
{
        [Key, MaxLen (256), Description (
          "  The name of the class or the subclass used in the "
          "creation of the System object in whose scope this "
          "PolicyCondition is defined.\n\n"
          "  "
          "This property helps to identify the System object in "
          "whose scope this instance of PolicyCondition exists. "
          "For a rule-specific PolicyCondition, this is the System "
          "in whose context the PolicyRule is defined.  For a "
          "reusable PolicyCondition, this is the instance of "
          "PolicyRepository (which is a subclass of System) that "
          "holds the Condition.\n\n"
          "  "
          "Note that this property, and the analogous property "
          "SystemName, do not represent propagated keys from an "
          "instance of the class System.  Instead, they are "
          "properties defined in the context of this class, which "
          "repeat the values from the instance of System to which "
          "this PolicyCondition is related, either directly via the "
          "PolicyConditionInPolicyRepository aggregation or indirectly "
          "via the PolicyConditionInPolicyRule aggregation.")
        ]
    string SystemCreationClassName;
        [Key, MaxLen (256), Description (
         "  The name of the System object in whose scope this "
         "PolicyCondition is defined.\n\n"
         "  "
         "This property completes the identification of the System "
         "object in whose scope this instance of PolicyCondition "
         "exists.  For a rule-specific PolicyCondition, this is the "
         "System in whose context the PolicyRule is defined.  For a "
         "reusable PolicyCondition, this is the instance of "
         "PolicyRepository (which is a subclass of System) that "
         "holds the Condition.")
        ]
    string SystemName;
        [Key, MaxLen (256), Description (
          "For a rule-specific PolicyCondition, the "
          "CreationClassName of the PolicyRule object with which "
          "this Condition is associated.  For a reusable Policy"
          "Condition, a special value, 'NO RULE', should be used to "
          "indicate that this Condition is reusable and not "
          "associated with a single PolicyRule.")
         ]
     string PolicyRuleCreationClassName;
         [Key, MaxLen (256), Description (
          "For a rule-specific PolicyCondition, the name of "
          "the PolicyRule object with which this Condition is "
          "associated.  For a reusable PolicyCondition, a "
          "special value, 'NO RULE', should be used to indicate "
          "that this Condition is reusable and not associated "
          "with a single PolicyRule.")
         ]
     string PolicyRuleName;
         [Key, MaxLen (256), Description (
            "CreationClassName indicates the name of the class or the "
            "subclass used in the creation of an instance.  When used "
            "with the other key properties of this class, this property "
            "allows all instances of this class and its subclasses to "
            "be uniquely identified.") ]
     string CreationClassName;
         [Key, MaxLen (256), Description (
            "A user-friendly name of this PolicyCondition.")
         ]
     string PolicyConditionName;
};

// ==================================================================
//    PolicyConditionInPolicyRule
// ==================================================================
   [Association, Aggregation, Description (
        "  A PolicyRule aggregates zero or more instances of the "
        "PolicyCondition class, via the PolicyConditionInPolicyRule "
        "association.  A Rule that aggregates zero Conditions is not "
        "valid -- it may, however, be in the process of being entered "
        "into a PolicyRepository or being defined for a System.  Note "
        "that a PolicyRule should have no effect until it is valid.\n\n"
        "  "
        "The Conditions aggregated by a PolicyRule are grouped into "
        "two levels of lists: either an ORed set of ANDed sets of "
        "conditions (DNF, the default) or an ANDed set of ORed sets "
        "of conditions (CNF).  Individual PolicyConditions in these "
        "lists may be negated.  The property ConditionListType "
        "specifies which of these two grouping schemes applies to a "
        "particular PolicyRule.\n\n"
        " "
        "In either case, PolicyConditions are used to determine whether "
        "to perform the PolicyActions associated with the
PolicyRule.\n\n"
        " "
        "One or more PolicyTimePeriodConditions may be among the "
        "conditions associated with a PolicyRule via the Policy"
        "ConditionInPolicyRule association.  In this case, the time "
        "periods are simply additional Conditions to be evaluated "
        "along with any others that are specified for the Rule. ")
   ]
class CIM_PolicyConditionInPolicyRule : CIM_PolicyComponent
{
        [Override ("GroupComponent"), Aggregate, Description (
         "This property represents the PolicyRule that "
         "contains one or more PolicyConditions.")
        ]
    CIM_PolicyRule REF GroupComponent;
        [Override ("PartComponent"), Description (
         "This property holds the name of a PolicyCondition "
         "contained by one or more PolicyRules.")
        ]
    CIM_PolicyCondition REF PartComponent;
        [Description (
         "Unsigned integer indicating the group to which the "
         "PolicyCondition identified by the ContainedCondition "
         "property belongs.  This integer segments the Conditions "
         "into the ANDed sets (when the ConditionListType is "
         "\"DNF\") or similarly the ORed sets (when the Condition"
         "ListType is \"CNF\") that are then evaluated.")
        ]
    uint16 GroupNumber;
        [Description (
         "Indication of whether the Condition identified by "
         "the ContainedCondition property is negated.  TRUE "
         "indicates that the PolicyCondition IS negated, FALSE "
         "indicates that it IS NOT negated.")
        ]
    boolean ConditionNegated;
};

// ==================================================================
//    PolicyConditionInPolicyRepository
// ==================================================================
   [Association, Description (
         "  A class representing the hosting of reusable "
         "PolicyConditions by a PolicyRepository.  A reusable Policy"
         "Condition is always related to a single PolicyRepository, "
```

```
          "via this aggregation.\n\n"
          "  "
          "Note, that an instance of PolicyCondition can be either "
          "reusable or rule-specific.  When the Condition is rule-"
          "specific, it shall not be related to any "
          "PolicyRepository via the PolicyConditionInPolicyRepository "
          "aggregation.")
   ]
class CIM_PolicyConditionInPolicyRepository : CIM_PolicyInSystem
{
        [Override ("Antecedent"), Max(1), Description (
         "This property identifies a PolicyRepository "
         "hosting one or more PolicyConditions.  A reusable "
         "PolicyCondition is always related to exactly one "
         "PolicyRepository via the PolicyConditionInPolicyRepository "
         "aggregation.  The [0..1] cardinality for this property "
         "covers the two types of PolicyConditions:  0 for a "
         "rule-specific PolicyCondition, 1 for a reusable one.")
        ]
    CIM_PolicyRepository REF Antecedent;
        [Override ("Dependent"), Description (
         "This property holds the name of a PolicyCondition"
         "hosted in the PolicyRepository. ")
        ]
    CIM_PolicyCondition REF Dependent;
};

// ==================================================================
// PolicyTimePeriodCondition
// ==================================================================
   [Description (
         "  This class provides a means of representing the time "
         "periods during which a PolicyRule is valid, i.e., active. "
         "At all times that fall outside these time periods, the "
         "PolicyRule has no effect.  A Rule is treated as valid "
         "at ALL times, if it does not specify a "
         "PolicyTimePeriodCondition.\n\n"
         "  "
         "In some cases a Policy Consumer may need to perform "
         "certain setup / cleanup actions when a PolicyRule becomes "
         "active / inactive.  For example, sessions that were "
         "established while a Rule was active might need to "
         "be taken down when the Rule becomes inactive.  In other "
         "cases, however, such sessions might be left up.  In this "
         "case, the effect of deactivating the PolicyRule would "
         "just be to prevent the establishment of new sessions. \n\n"
         "  "
         "Setup / cleanup behaviors on validity period "
        "transitions are not currently addressed by the Policy "
        "Model, and must be specified in 'guideline' documents or "
        "via subclasses of CIM_PolicyRule, CIM_PolicyTimePeriod"
        "Condition or other concrete subclasses of CIM_Policy.  If "
        "such behaviors need to be under the control of the policy "
        "administrator, then a mechanism to allow this control "
        "must also be specified in the subclasses.\n\n"
        "  "
        "PolicyTimePeriodCondition is defined as a subclass of "
        "PolicyCondition.  This is to allow the inclusion of "
        "time-based criteria in the AND/OR condition definitions "
        "for a PolicyRule.\n\n"
        "  "
        "Instances of this class may have up to five properties "
        "identifying time periods at different levels.  The values "
        "of all the properties present in an instance are ANDed "
        "together to determine the validity period(s) for the "
        "instance.  For example, an instance with an overall "
        "validity range of January 1, 2000 through December 31, "
        "2000; a month mask that selects March and April; a "
        "day-of-the-week mask that selects Fridays; and a time "
        "of day range of 0800 through 1600 would be represented "
        "using the following time periods:\n"
        "   Friday, March  5, 2000, from 0800 through 1600;\n "
        "   Friday, March 12, 2000, from 0800 through 1600;\n "
        "   Friday, March 19, 2000, from 0800 through 1600;\n "
        "   Friday, March 26, 2000, from 0800 through 1600;\n "
        "   Friday, April  2, 2000, from 0800 through 1600;\n "
        "   Friday, April  9, 2000, from 0800 through 1600;\n "
        "   Friday, April 16, 2000, from 0800 through 1600;\n "
        "   Friday, April 23, 2000, from 0800 through 1600;\n "
        "   Friday, April 30, 2000, from 0800 through 1600.\n\n"
        "  "
        "Properties not present in an instance of "
        "PolicyTimePeriodCondition are implicitly treated as having "
        "their value 'always enabled'.  Thus, in the example above, "
        "the day-of-the-month mask is not present, and so the "
        "validity period for the instance implicitly includes a "
        "day-of-the-month mask that selects all days of the month. "
        "If this 'missing property' rule is applied to its fullest, we "
        "see that there is a second way to indicate that a Policy"
        "Rule is always enabled: associate with it an instance of "
        "PolicyTimePeriodCondition whose only properties with "
        "specific values are its key properties.")
    ]
class CIM_PolicyTimePeriodCondition : CIM_PolicyCondition
{
        [Description (
        "  This property identifies an overall range of calendar "
        "dates and times over which a PolicyRule is valid.  It is "
        "formatted as a string representing a start date and time, "
        "in which the character 'T' indicates the beginning of the "
        "time portion, followed by the solidus character '/', "
        "followed by a similar string representing an end date and "
        "time.  The first date indicates the beginning of the range, "
        "while the second date indicates the end.  Thus, the second "
        "date and time must be later than the first.  Date/times are "
        "expressed as substrings of the form yyyymmddThhmmss.  For "
        "example: \n"
        "   20000101T080000/20000131T120000 defines \n"
        "   January 1, 2000, 0800 through January 31, 2000, noon\n\n"
        "  "
        "There are also two special cases in which one of the "
        "date/time strings is replaced with a special string defined "
        "in RFC 2445.\n "
        "   o If the first date/time is replaced with the string "
        "     'THISANDPRIOR', then the property indicates that a "
        "     PolicyRule is valid [from now] until the date/time "
        "     that appears after the '/'.\n"
        "   o If the second date/time is replaced with the string "
        "     'THISANDFUTURE', then the property indicates that a "
        "     PolicyRule becomes valid on the date/time that "
        "     appears before the '/', and remains valid from that "
        "     point on. "),
        ModelCorrespondence {
       "CIM_PolicyTimePeriodCondition.MonthOfYearMask",
       "CIM_PolicyTimePeriodCondition.DayOfMonthMask",
       "CIM_PolicyTimePeriodCondition.DayOfWeekMask",
       "CIM_PolicyTimePeriodCondition.TimeOfDayMask",
       "CIM_PolicyTimePeriodCondition.LocalOrUtcTime"}
       ]
   string TimePeriod;
       [Octetstring, Description (
        "  The purpose of this property is to refine the valid time "
        "period that is defined by the TimePeriod property, by "
        "explicitly specifying in which months the PolicyRule is "
        "valid.  These properties work together, with the "
        "TimePeriod used to specify the overall time period in "
        "which the PolicyRule is valid, and the MonthOfYearMask used "
        "to pick out the months during which the Rule is valid.\n\n"
        "  "
        "This property is formatted as an octet string, structured "
        "as follows:\n"
        "   o a 4-octet length field, indicating the length of the "
        "    entire octet string; this field is always set to "
        "    0x00000006 for this property;\n"
```

```
         "   o a 2-octet field consisting of 12 bits identifying the "
         "     12 months of the year, beginning with January and "
         "     ending with December, followed by 4 bits that are "
         "     always set to '0'.  For each month, the value '1' "
         "     indicates that the policy is valid for that month, "
         "     and the value '0' indicates that it is not valid.\n\n"
         "  "
         "The value 0x000000060830, for example, indicates that a "
         "PolicyRule is valid only in the months May, November, "
         "and December.\n\n"
         "  "
         "If a value for this property is not provided, then the "
         "PolicyRule is treated as valid for all twelve months, and "
         "only restricted by its TimePeriod property value and the "
         "other Mask properties."),
        ModelCorrespondence {
        "CIM_PolicyTimePeriodCondition.TimePeriod",
        "CIM_PolicyTimePeriodCondition.LocalOrUtcTime"}
        ]
    uint8 MonthOfYearMask[];
        [Octetstring, Description (
         "  The purpose of this property is to refine the valid time "
         "period that is defined by the TimePeriod property, by "
         "explicitly specifying in which days of the month the Policy"
         "Rule is valid.  These properties work together, "
         "with the TimePeriod used to specify the overall time period "
         "in which the PolicyRule is valid, and the DayOfMonthMask used "
         "to pick out the days of the month during which the Rule "
         "is valid.\n\n "
         "  "
         "This property is formatted as an octet string, structured "
         "as follows:\n"
         "   o a 4-octet length field, indicating the length of the "
         "     entire octet string; this field is always set to "
         "     0x0000000C for this property; \n"
         "   o an 8-octet field consisting of 31 bits identifying "
         "     the days of the month counting from the beginning, "
         "     followed by 31 more bits identifying the days of the "
         "     month counting from the end, followed by 2 bits that "
         "     are always set to '0'.  For each day, the value '1' "
         "     indicates that the policy is valid for that day, and "
         "     the value '0' indicates that it is not valid. \n\n"
         "  "
         "The value 0x0000000C8000000100000000, for example, "
         "indicates that a PolicyRule is valid on the first and "
         "last days of the month.\n\n "
         "  "
         "For months with fewer than 31 days, the digits corresponding "
```

```
           "to days that the months do not have (counting in both "
           "directions) are ignored.\n\n"
           " "
           "If a value for this property is not provided, then the "
           "PolicyRule is treated as valid for all days of the month, and "
           "only restricted by its TimePeriod property value and the "
           "other Mask properties."),
          ModelCorrespondence {
          "CIM_PolicyTimePeriodCondition.TimePeriod",
          "CIM_PolicyTimePeriodCondition.LocalOrUtcTime"}
          ]
      uint8 DayOfMonthMask[];
          [Octetstring, Description (
           "  The purpose of this property is to refine the valid time "
           "period that is defined by the TimePeriod property, by "
           "explicitly specifying in which days of the month the Policy"
           "Rule is valid.  These properties work together, "
           "with the TimePeriod used to specify the overall time period "
           "in which the PolicyRule is valid, and the DayOfWeekMask used "
           "to pick out the days of the week during which the Rule "
           "is valid.\n\n "
           " "
           "This property is formatted as an octet string, structured "
           "as follows:\n "
           "  o a 4-octet length field, indicating the length of the "
           "    entire octet string; this field is always set to "
           "    0x00000005 for this property;\n"
           "  o a 1-octet field consisting of 7 bits identifying the 7 "
           "    days of the week, beginning with Sunday and ending with "
           "    Saturday, followed by 1 bit that is always set to '0'. "
           "    For each day of the week, the value '1' indicates that "
           "    the policy is valid for that day, and the value '0' "
           "    indicates that it is not valid. \n\n"
           " "
           "The value 0x000000057C, for example, indicates that a "
           "PolicyRule is valid Monday through Friday.\n\n"
           " "
           "If a value for this property is not provided, then the "
           "PolicyRule is treated as valid for all days of the week, "
           "and only restricted by its TimePeriod property value and "
           "the other Mask properties."),
          ModelCorrespondence {
          "CIM_PolicyTimePeriodCondition.TimePeriod",
          "CIM_PolicyTimePeriodCondition.LocalOrUtcTime"}
          ]
      uint8 DayOfWeekMask[];
          [Description (
           "  The purpose of this property is to refine the valid time "
```

```
         "period that is defined by the TimePeriod property, by "
         "explicitly specifying a range of times in a day during which "
         "the PolicyRule is valid.  These properties work "
         "together, with the TimePeriod used to specify the overall "
         "time period in which the PolicyRule is valid, and the "
         "TimeOfDayMask used to pick out the range of time periods "
         "in a given day of during which the Rule is valid. \n\n"
         "  "
         "This property is formatted in the style of RFC 2445:  a "
         "time string beginning with the character 'T', followed by "
         "the solidus character '/', followed by a second time string. "
         "The first time indicates the beginning of the range, while "
         "the second time indicates the end.  Times are expressed as "
         "substrings of the form 'Thhmmss'. \n\n"
         "  "
         "The second substring always identifies a later time than "
         "the first substring.  To allow for ranges that span "
         "midnight, however, the value of the second string may be "
         "smaller than the value of the first substring.  Thus, "
         "'T080000/T210000' identifies the range from 0800 until 2100, "
         "while 'T210000/T080000' identifies the range from 2100 until "
         "0800 of the following day. \n\n"
         "  "
         "When a range spans midnight, it by definition includes "
         "parts of two successive days.  When one of these days is "
         "also selected by either the MonthOfYearMask, "
         "DayOfMonthMask, and/or DayOfWeekMask, but the other day is "
         "not, then the policy is active only during the portion of "
         "the range that falls on the selected day.  For example, if "
         "the range extends from 2100 until 0800, and the day of "
         "week mask selects Monday and Tuesday, then the policy is "
         "active during the following three intervals:\n"
         "    From midnight Sunday until 0800 Monday; \n"
         "    From 2100 Monday until 0800 Tuesday; \n"
         "    From 2100 Tuesday until 23:59:59 Tuesday. \n\n"
         "  "
         "If a value for this property is not provided, then the "
         "PolicyRule is treated as valid for all hours of the day, "
         "and only restricted by its TimePeriod property value and "
         "the other Mask properties."),
        ModelCorrespondence {
        "CIM_PolicyTimePeriodCondition.TimePeriod",
        "CIM_PolicyTimePeriodCondition.LocalOrUtcTime"}
        ]
    string TimeOfDayMask;
        [Description (
         "  This property indicates whether the times represented "
         "in the TimePeriod property and in the various Mask "
```

```
          "properties represent local times or UTC times.  There is "
          "no provision for mixing of local times and UTC times:  the "
          "value of this property applies to all of the other "
          "time-related properties."),
          ValueMap { "1", "2" },
          Values { "localTime", "utcTime" },
          ModelCorrespondence {
          "CIM_PolicyTimePeriodCondition.TimePeriod",
          "CIM_PolicyTimePeriodCondition.MonthOfYearMask",
          "CIM_PolicyTimePeriodCondition.DayOfMonthMask",
          "CIM_PolicyTimePeriodCondition.DayOfWeekMask",
          "CIM_PolicyTimePeriodCondition.TimeOfDayMask"}
         ]
     uint16 LocalOrUtcTime;
};

// ==================================================================
//    PolicyRuleValidityPeriod
// ==================================================================
   [Association, Aggregation, Description (
         "The PolicyRuleValidityPeriod aggregation represents "
         "scheduled activation and deactivation of a PolicyRule. "
         "If a PolicyRule is associated with multiple policy time "
         "periods via this association, then the Rule is active if "
         "at least one of the time periods indicates that it is "
         "active.  (In other words, the PolicyTimePeriodConditions "
         "are ORed to determine whether the Rule is active.)  A Time"
         "Period may be aggregated by multiple PolicyRules.  A Rule "
         "that does not point to a PolicyTimePeriodCondition via this "
         "association is, from the point of view of scheduling, "
         "always active.  It may, however, be inactive for other "
         "reasons.  For example, the Rule's Enabled property may "
         "be set to \"disabled\" (value=2).")
   ]
class CIM_PolicyRuleValidityPeriod : CIM_PolicyComponent
{
        [Override ("GroupComponent"), Aggregate, Description (
         "This property contains the name of a PolicyRule that "
         "contains one or more PolicyTimePeriodConditions.")
        ]
    CIM_PolicyRule REF GroupComponent;
        [Override ("PartComponent"), Description (
         "This property contains the name of a "
         "PolicyTimePeriodCondition defining the valid time periods "
         "for one or more PolicyRules.")
        ]
    CIM_PolicyTimePeriodCondition REF PartComponent;
};
```

```
// ==================================================================
// VendorPolicyCondition
// ==================================================================
   [Description (
         "  A class that provides a general extension mechanism for "
         "representing PolicyConditions that have not been modeled "
         "with specific properties.  Instead, the two properties "
         "Constraint and ConstraintEncoding are used to define the "
         "content and format of the Condition, as explained below.\n\n"
         "  "
         "As its name suggests, VendorPolicyCondition is intended for "
         "vendor-specific extensions to the Policy Core Information "
         "Model.  Standardized extensions are not expected to use "
         "this class.")
   ]
class CIM_VendorPolicyCondition : CIM_PolicyCondition
{
        [Octetstring, Description (
         "This property provides a general extension mechanism for "
         "representing PolicyConditions that have not been "
         "modeled with specific properties.  The format of the "
         "octet strings in the array is left unspecified in "
         "this definition.  It is determined by the OID value "
         "stored in the property ConstraintEncoding.  Since "
         "ConstraintEncoding is single-valued, all the values of "
         "Constraint share the same format and semantics."),
         ModelCorrespondence {
            "CIM_VendorPolicyCondition.ConstraintEncoding"}
        ]
    string Constraint [];
        [Description (
         "An OID encoded as a string, identifying the format "
         "and semantics for this instance's Constraint property."),
         ModelCorrespondence {
            "CIM_VendorPolicyCondition.Constraint"}
        ]
    string ConstraintEncoding;
};

// ==================================================================
// PolicyAction
// ==================================================================
   [Abstract, Description (
         "A class representing a rule-specific or reusable policy "
         "action to be performed if the PolicyConditions for a Policy"
         "Rule evaluate to TRUE.  Since all operational details of a "
         "PolicyAction are provided in subclasses of this object, "
         "this class is abstract.")
   ]
class CIM_PolicyAction : CIM_Policy
{
        [Key, MaxLen (256), Description (
         "  The name of the class or the subclass used in the "
         "creation of the System object in whose scope this "
         "PolicyAction is defined. \n\n"
         "  "
         "This property helps to identify the System object in "
         "whose scope this instance of PolicyAction exists. "
         "For a rule-specific PolicyAction, this is the System "
         "in whose context the PolicyRule is defined.  For a "
         "reusable PolicyAction, this is the instance of "
         "PolicyRepository (which is a subclass of System) that "
         "holds the Action. \n\n"
         "  "
         "Note that this property, and the analogous property "
         "SystemName, do not represent propagated keys from an "
         "instance of the class System.  Instead, they are "
         "properties defined in the context of this class, which "
         "repeat the values from the instance of System to which "
         "this PolicyAction is related, either directly via the "
         "PolicyActionInPolicyRepository aggregation or indirectly "
         "via the PolicyActionInPolicyRule aggregation.")
        ]
    string SystemCreationClassName;
        [Key, MaxLen (256), Description (
         "  The name of the System object in whose scope this "
         "PolicyAction is defined. \n\n"
         "  "
         "This property completes the identification of the System "
         "object in whose scope this instance of PolicyAction "
         "exists.  For a rule-specific PolicyAction, this is the "
         "System in whose context the PolicyRule is defined.  For "
         "a reusable PolicyAction, this is the instance of "
         "PolicyRepository (which is a subclass of System) that "
         "holds the Action.")
        ]
    string SystemName;
        [Key, MaxLen (256), Description (
         "For a rule-specific PolicyAction, the CreationClassName "
         "of the PolicyRule object with which this Action is "
         "associated.  For a reusable PolicyAction, a "
         "special value, 'NO RULE', should be used to "
         "indicate that this Action is reusable and not "
         "associated with a single PolicyRule.")
        ]
    string PolicyRuleCreationClassName;
```

```
        [Key, MaxLen (256), Description (
         "For a rule-specific PolicyAction, the name of "
         "the PolicyRule object with which this Action is "
         "associated.  For a reusable PolicyAction, a "
         "special value, 'NO RULE', should be used to "
         "indicate that this Action is reusable and not "
         "associated with a single PolicyRule.")
        ]
    string PolicyRuleName;
        [Key, MaxLen (256), Description (
           "CreationClassName indicates the name of the class or the "
           "subclass used in the creation of an instance.  When used "
           "with the other key properties of this class, this property "
           "allows all instances of this class and its subclasses to "
           "be uniquely identified.") ]
    string CreationClassName;
        [Key, MaxLen (256), Description (
         "A user-friendly name of this PolicyAction.")
        ]
    string PolicyActionName;
};

// ==================================================================
//    PolicyActionInPolicyRepository
// ==================================================================
   [Association, Description (
         "  A class representing the hosting of reusable "
         "PolicyActions by a PolicyRepository.  A reusable Policy"
         "Action is always related to a single PolicyRepository, "
         "via this aggregation.\n\n"
         "  "
         "Note, that an instance of PolicyAction can be either "
         "reusable or rule-specific.  When the Action is rule-"
         "specific, it shall not be related to any "
         "PolicyRepository via the PolicyActionInPolicyRepository "
         "aggregation.")
   ]
class CIM_PolicyActionInPolicyRepository : CIM_PolicyInSystem
{
        [Override ("Antecedent"), Max(1), Description (
         "This property represents a PolicyRepository "
         "hosting one or more PolicyActions.  A reusable "
         "PolicyAction is always related to exactly one "
         "PolicyRepository via the PolicyActionInPolicyRepository "
         "aggregation.  The [0..1] cardinality for this property "
         "covers the two types of PolicyActions:  0 for a "
         "rule-specific PolicyAction, 1 for a reusable one.")
        ]
        CIM_PolicyRepository REF Antecedent;
            [Override ("Dependent"), Description (
             "This property holds the name of a PolicyAction"
             "hosted in the PolicyRepository. ")
            ]
        CIM_PolicyAction REF Dependent;
    };

    // ==================================================================
    //    PolicyActionInPolicyRule
    // ==================================================================
       [Association, Aggregation, Description (
            "  A PolicyRule aggregates zero or more instances of the "
            "PolicyAction class, via the PolicyActionInPolicyRule "
            "association.  A Rule that aggregates zero Actions is not "
            "valid -- it may, however, be in the process of being entered "
            "into a PolicyRepository or being defined for a System. "
            "Alternately, the actions of the policy may be explicit in "
            "the definition of the PolicyRule.  Note that a PolicyRule "
            "should have no effect until it is valid.\n\n"
            "  "
            "The Actions associated with a PolicyRule may be given a "
            "required order, a recommended order, or no order at all.  For "
            "Actions represented as separate objects, the PolicyActionIn"
            "PolicyRule aggregation can be used to express an order. \n\n"
            "  "
            "This aggregation does not indicate whether a specified "
            "action order is required, recommended, or of no significance; "
            "the property SequencedActions in the aggregating instance of "
            "PolicyRule provides this indication.")
       ]
    class CIM_PolicyActionInPolicyRule : CIM_PolicyComponent
    {
            [Override ("GroupComponent"), Aggregate, Description (
             "This property represents the PolicyRule that "
             "contains one or more PolicyActions.")
            ]
        CIM_PolicyRule REF GroupComponent;
            [Override ("PartComponent"), Description (
             "This property holds the name of a PolicyAction "
             "contained by one or more PolicyRules.")
            ]
        CIM_PolicyAction REF PartComponent;
            [Description (
             "  This property provides an unsigned integer 'n' that"
             "indicates the relative position of a PolicyAction in the "
             "sequence of actions associated with a PolicyRule. "
             "When 'n' is a positive integer, it indicates a place "
```

```
          "in the sequence of actions to be performed, with "
          "smaller integers indicating earlier positions in the "
          "sequence.  The special value '0' indicates 'don't care'. "
          "If two or more PolicyActions have the same non-zero "
          "sequence number, they may be performed in any order, but "
          "they must all be performed at the appropriate place in the "
          "overall action sequence. \n\n"
          "  "
          "A series of examples will make ordering of PolicyActions "
          "clearer: \n"
          "   o If all actions have the same sequence number, "
          "     regardless of whether it is '0' or non-zero, any "
          "     order is acceptable.\n "
          "   o The values: \n"
          "         1:ACTION A \n"
          "         2:ACTION B \n"
          "         1:ACTION C \n"
          "         3:ACTION D \n"
          "     indicate two acceptable orders: A,C,B,D or C,A,B,D, "
          "     since A and C can be performed in either order, but "
          "     only at the '1' position. \n"
          "   o The values: \n"
          "         0:ACTION A \n"
          "         2:ACTION B \n"
          "         3:ACTION C \n"
          "         3:ACTION D \n"
          "     require that B,C, and D occur either as B,C,D or as "
          "     B,D,C.  Action A may appear at any point relative to "
          "     B, C, and D.  Thus the complete set of acceptable "
          "     orders is:  A,B,C,D; B,A,C,D; B,C,A,D; B,C,D,A; "
          "     A,B,D,C; B,A,D,C; B,D,A,C; B,D,C,A. \n\n"
          "  "
          "Note that the non-zero sequence numbers need not start "
          "with '1', and they need not be consecutive.  All that "
          "matters is their relative magnitude.")
        ]
    uint16 ActionOrder;
};

// ==================================================================
// VendorPolicyAction
// ==================================================================
   [Description (
          "  A class that provides a general extension mechanism for "
          "representing PolicyActions that have not been modeled "
          "with specific properties.  Instead, the two properties "
          "ActionData and ActionEncoding are used to define the "
          "content and format of the Action, as explained below.\n\n"
```

```
         "  "
         "As its name suggests, VendorPolicyAction is intended for "
         "vendor-specific extensions to the Policy Core Information "
         "Model.  Standardized extensions are not expected to use "
         "this class.")  ]
class CIM_VendorPolicyAction : CIM_PolicyAction
{
        [Octetstring, Description (
         "This property provides a general extension mechanism for "
         "representing PolicyActions that have not been "
         "modeled with specific properties.  The format of the "
         "octet strings in the array is left unspecified in "
         "this definition.  It is determined by the OID value "
         "stored in the property ActionEncoding.  Since "
         "ActionEncoding is single-valued, all the values of "
         "ActionData share the same format and semantics."),
         ModelCorrespondence {
            "CIM_VendorPolicyAction.ActionEncoding"}
        ]
    string ActionData [];
        [Description (
         "An OID encoded as a string, identifying the format "
         "and semantics for this instance's ActionData property."),
         ModelCorrespondence {
            "CIM_VendorPolicyAction.ActionData"}
        ]
    string ActionEncoding;
};

// ==================================================================
// end of file
// ==================================================================
```

15. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.